21世纪高职高专规划教材

软件专业系列

案例式教材



软件测试教程

杜文洁 主编



清华大学出版社

21 世纪高职高专规划教材·软件专业系列

软件测试教程

杜文洁　主编

清华大学出版社
北　京

内 容 简 介

本书详尽地阐述了软件测试的基础知识及其相关的实用技术。内容包括软件测试概述、软件测试过程与策略、黑盒测试及其实例设计、白盒测试及其实例设计、软件测试计划与文档、软件自动化测试、软件测试管理、面向对象的软件测试、Web 网站测试及软件测试职业。

本书结合教学实例突出基本知识和基本概念的表述，注重内容的先进性、系统性和实用性，力求反映软件测试发展的最新成果。将测试与软件工程密切结合，使读者可以更好地理解和掌握软件测试的内容，并迅速地运用到实际测试工作中去。

本书可作为高等院校、高职高专院校及相关软件学院软件技术专业和计算机相关专业的教材，也可作为软件测试技术培训的教材，同时还可供从事软件测试的工作人员参阅。

图书在版编目(CIP)数据

软件测试教程/杜文洁主编. —北京：清华大学出版社，2008.4
21 世纪高职高专规划教材. 软件专业系列
ISBN 978-7-302-16788-4

Ⅰ. 软…　Ⅱ. 杜…　Ⅲ. 软件—测试—高等学校：技术学校—教材　Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字(2008)第 004136 号

责任编辑：束传政　田　梅
责任校对：袁　芳
责任印制：

出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：185×230　**印　张**：14.25　**字　数**：292 千字
版　次：2008 年 4 月第 1 版　**印　次**：2008 年 4 月第1 次印刷
印　数：1～　000
定　价：　.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：024865-01

出版说明

高职高专教育是我国高等教育的重要组成部分，担负着为国家培养并输送生产、建设、管理、服务第一线高素质技术应用型人才的重任。

进入21世纪后，高职高专教育的改革和发展呈现出前所未有的发展势头，学生规模已占我国高等教育的半壁江山，成为我国高等教育的一支重要的生力军；办学理念上，“以就业为导向”成为高等职业教育改革与发展的主旋律。近两年来，教育部召开了三次产学研交流会，并启动四个专业的“国家技能型紧缺人才培养项目”，同时成立了35所示范性软件职业技术学院，进行两年制教学改革试点。这些举措都表明国家正在推动高职高专教育进行深层次的重大改革，向培养生产、服务第一线真正需要的应用型人才的方向发展。

为了顺应当前我国高职高专教育的发展形势，配合高职高专院校的教学改革和教材建设，进一步提高我国高职高专教育教材质量，在教育部的指导下，清华大学出版社组织出版了“21世纪高职高专规划教材”。

为推动规划教材的建设，清华大学出版社组织并成立了“高职高专教育教材编审委员会”，旨在对清华版的全国性高职高专教材及教材选题进行评审，并向清华大学出版社推荐各院校办学特色鲜明、内容质量优秀的教材选题。教材选题由个人或各院校推荐，经编审委员会认真评审，最后由清华大学出版社出版。编审委员会的成员皆来源于教改成效大、办学特色鲜明、师资实力强的高职高专院校、普通高校以及著名企业，教材的编写者和审定者都是从事高职高专教育第一线的骨干教师和专家。

编审委员会根据教育部最新文件和政策，规划教材体系，比如部分专业的两年制教材；“以就业为导向”，以“专业技能体系”为主，突出人才培养的实践性、应用性的原则，重新组织系列课程的教材结构，整合课程体系；按照教育部制定的“高职高专教育基础课程教学基本要求”，教材的基础理论以“必要、够用”为度，突出基础理论的应用和实践技能的培养。

本套规划教材的编写原则如下：

(1) 根据岗位群设置教材系列，并成立系列教材编审委员会；

(2) 由编审委员会规划教材、评审教材；

(3) 重点课程进行立体化建设，突出案例式教学体系，加强实训教材的出版，完善教学服务体系；

(4) 教材编写者由具有丰富教学经验和多年实践经历的教师共同组成，建立“双师

型”编者体系。

本套规划教材涵盖了公共基础课、计算机、电子信息、机械、经济管理以及服务等大类的主要课程，包括专业基础课和专业主干课。目前已经规划的教材系列名称如下：

• 公共基础课

公共基础课系列

• 计算机类

计算机基础教育系列
计算机专业基础系列
计算机应用系列
网络专业系列
软件专业系列
电子商务专业系列

• 电子信息类

电子信息基础系列
微电子技术系列
通信技术系列
电气、自动化、应用电子技术系列

• 机械类

机械基础系列
机械设计与制造专业系列
数控技术系列
模具设计与制造系列

• 经济管理类

经济管理基础系列
市场营销系列
财务会计系列
企业管理系列
物流管理系列
财政金融系列
国际商务系列

• 服务类

艺术设计系列

本套规划教材的系列名称根据学科基础和岗位群方向设置，为各高职高专院校提供“自助餐”形式的教材。各院校在选择课程需要的教材时，专业课程可以根据岗位群选择系列；专业基础课程可以根据学科方向选择各类的基础课系列。例如，数控技术方向的专业课程可以在“数控技术系列”选择；数控技术专业需要的基础课程，属于计算机类课程的可以在“计算机基础教育系列”和“计算机应用系列”选择，属于机械类课程的可以在“机械基础系列”选择，属于电子信息类课程的可以在“电子信息基础系列”选择。依此类推。

为方便教师授课和学生学习，清华大学出版社正在建设本套教材的教学服务体系。本套教材先期选择重点课程和专业主干课程，进行立体化教材建设：加强多媒体教学课件或电子教案、素材库、学习盘、学习指导书等形式的制作和出版，开发网络课程。学校在选用教材时，可通过邮件或电话与我们联系获取相关服务，并通过与各院校的密切交流，使其日臻完善。

高职高专教育正处于新一轮改革时期，从专业设置、课程体系建设到教材编写，依然是新课题。希望各高职高专院校在教学实践中积极提出意见和建议，并向我们推荐优秀选题。反馈意见请发送到 E-mail：gzgz@tup.tsinghua.edu.cn。清华大学出版社将对已出版的教材不断地修订、完善，提高教材质量，完善教材服务体系，为我国的高职高专教育出版优秀的高质量的教材。

高职高专教育教材编审委员会

前　言

软件测试是对软件需求分析、设计规格说明和编码的最终审核，是软件质量保证的关键步骤。随着软件产业的迅速发展，市场对于进行专业化、高效化软件测试的需求越来越强烈，软件测试职业的价值越发显著，软件测试技术作为一门新兴产业迅速发展起来。在这种形势下，计算机专业学生对于软件测试知识的学习与应用就显得尤为重要，因此一本好的软件测试教材对于学习者是必不可少的。

现阶段国内软件测试教学处于起步阶段，我们依据高职高专软件测试课程教学大纲所规定的教学要求编写本教材，把多年软件测试教学经验和教学实践成果融入本教材中，在内容分布上充分考虑理论与实践相结合的原则。

本书选材注意把握高职高专学生的专业知识背景与接受能力，由浅入深，以“实例教学”的方法激发学生的学习兴趣。在编写上，注重内容的先进性、系统性和实用性，力求反映软件测试发展的最新成果。在结构安排上，深入阐述软件测试的基础理论知识，循序渐进，做到了理论和实际相结合。在每章内容后面均附有小结和课后习题。

本书共分 10 章，系统地介绍了软件测试的主要内容，具体如下：

1. 软件测试概述

介绍了软件测试的背景、基本理论，以及软件测试与软件开发的关系。

2. 软件测试过程与策略

介绍了软件测试的复杂性与经济性分析，描述了软件测试的流程，其中包括单元测试、集成测试、确认测试、系统测试和验收测试 5 个测试阶段。最后，介绍了两组测试方法，分别是静态测试与动态测试、黑盒测试与白盒测试。

3. 黑盒测试及其实例设计

具体介绍了黑盒测试的各种方法，其中包括等价类划分法、边界值分析法、决策表法、因果图法，并且分别配有测试用例来具体说明这 4 种方法。最后描述了黑盒测试的一个实例设计。

4. 白盒测试及其实例设计

具体介绍了白盒测试的各种方法，其中包括逻辑覆盖测试和路径分析测试两种主要

方法。这部分还介绍了循环测试、变异测试和程序插装等其他白盒测试方法。最后，列举了一个采用覆盖测试方法的测试实例和一个采用独立路径测试方法的测试实例。

5. 软件测试计划与文档

详细阐述了测试计划的制定、测试文档的主要内容和软件生命周期各阶段的测试任务与可交付的文档，列举了测试用例、测试总结报告的设计内容。

6. 软件自动化测试

介绍了软件自动化测试、自动化测试的设计与开发以及常用的自动化测试工具。

7. 软件测试管理

具体叙述了软件质量保证与软件测试的关系、测试的组织管理和测试团队的职责，以及 ISO 9000 标准和能力成熟度模型(CMM)。

8. 面向对象的软件测试

阐述了面向对象的软件测试、面向对象的软件、面向对象测试与传统测试的区别和面向对象的测试方法。

9. Web 网站测试

阐述了 Web 网站的测试、功能测试、性能测试、安全性测试、可用性/可靠性测试、配置和兼容性测试、数据库测试。

10. 软件测试职业

介绍了软件测试职业和职位、软件测试资源的获取途径和软件测试工程师的素质要求。

本书由杜文洁担任主编，景秀丽、白萍担任副主编。第 1 章、第 10 章由杜文洁编写，第 2 章、第 4 章、第 5 章、第 7 章、第 9 章由景秀丽编写，第 3 章、第 6 章、第 8 章由白萍编写。全书由杜文洁统稿完成。

由于水平和时间有限，书中难免存在一些疏漏，请读者批评指正。

作　者

2008 年 3 月

目　录

第 1 章

软件测试概述

本章概述

本章介绍了软件测试的发展历史，软件测试技术的分类方法、测试标准、测试原则，阐述了软件测试与软件开发的关系。

1.1 软件测试的背景

软件的质量就是软件的生命，为了保证软件的质量，人们在长期的开发过程中积累了许多经验并形成了许多行之有效的方法。但是借助这些方法，我们只能尽量减少软件中的错误和不足，却不能完全避免所有的错误。

由于软件是人脑的高度智力化的体现和产品这一特殊性，不同于其他科技和生产领域，因此软件与生俱来就有可能存在着缺陷。

在开发大型软件系统的漫长过程中，面对纷繁复杂的各种现实情况，人的主观认识和客观现实之间往往存在着差距，开发过程中的各类人员之间的交流和配合也往往并不是尽善尽美的。

如果我们不能在软件正式投入运行之前发现并纠正这些错误，那么这些错误最终必然会在软件的实际运行过程中暴露出来。到那时，不仅改正这些错误要付出很大的代价，而且往往会造成无法弥补的损失。

如何防止和减少这些可能存在的问题呢？回答是进行软件测试。测试是最有效的排除和防止软件缺陷与故障的手段，并由此促进了软件测试理论与技术实践的快速发展。新的测试理论、测试方法、测试技术手段在不断涌出，软件测试机构和组织也在迅速产生和发展，由此软件测试技术职业也同步完善和健全起来。

1.1.1 软件缺陷

1. 软件错误案例研究

人们常常不把软件当回事，没有真正意识到它已经深入渗透到我们的日常生活中，软件在电子信息领域里无处不在。现在有许多人如果一天不上网查看电子邮件，简直就没法过下去。我们已经离不开24小时包裹投递服务、长途电话服务和最先进的医疗服务了。

然而软件是由人编写开发的，是一种逻辑思维的产品，尽管现在软件开发者采取了一系列有效措施，不断地提高软件开发质量，但仍然无法完全避免软件（产品）会存在各种各样的缺陷。

下面以实例来说明。

（1）迪斯尼的狮子王游戏软件缺陷

1994年秋天，迪斯尼公司发布了第一个面向儿童的多媒体光盘游戏——狮子王动画故事书（The Lion King Animated Storybook）。尽管当时已经有许多其他公司在儿童游戏市场上运作多年，但是这次是迪斯尼公司首次进军儿童游戏市场，所以进行了大量促销宣传。结果，销售额非常可观，该游戏成为孩子们在当年节假日的“必买游戏”。然而后来却飞来横祸。1994年12月26日，圣诞节过后的第一天，迪斯尼公司的客户支持电话开始响个不停。很快，电话支持技术员们就淹没在来自于愤怒的家长并伴随着玩不成游戏的孩子们哭叫的电话之中。报纸和电视新闻进行了大量的报道。

后来证实，迪斯尼公司未能对市面上投入使用的许多不同类型的PC机型进行广泛的测试。软件只能在极少数系统中工作正常——例如在迪斯尼程序员用来开发游戏的系统中——但在大多数公众使用的系统中却不能运行。

（2）爱国者导弹防御系统缺陷

爱国者导弹防御系统是里根总统提出的战略防御计划（即星球大战计划）的缩略版本，它首次应用在海湾战争中对抗伊拉克飞毛腿导弹的防御战中。尽管对系统赞誉的报道不绝于耳，但是它确实在对抗几枚导弹中失利，包括一次在沙特阿拉伯的多哈击毙了28名美国士兵。分析发现症结在于一个软件缺陷，系统时钟的一个很小的计时错误积累起来到14小时后，跟踪系统不再准确。在多哈的这次袭击中，系统已经运行了100多个小时。

（3）千年虫问题

20世纪70年代早期的某个时间，某位程序员正在为本公司设计开发工资系统。他使用的计算机存储空间很小，迫使他尽量节省每一个字节。他将自己的程序压缩得比其他任何人都紧凑。使用的其中一个方法是把4位数年份，例如1973年，缩减为2位数，73。因为工资系统相当依赖于日期的处理，所以需要节省大量的存储空间。他简单地认

为只有在到达 2000 年，那时他的程序开始计算 00 或 01 这样的年份时问题才会产生。虽然他知道会出这样的问题，但是他认定在 25 年之内程序肯定会升级或替换，而且眼前的任务比现在计划遥不可及的未来更加重要。然而这一天毕竟到来了。1995 年他的程序仍然在使用，而他退休了，谁也不会想到如何深入到程序中检查 2000 年兼容问题，更不用说去修改了。

估计全球各地更换或升级类似的前者程序以解决潜在的 2000 年问题的费用已经达数千亿美元。

(4) 美国航天局火星登陆探测器缺陷

1999 年 12 月 3 日，美国航天局的火星极地登陆者号探测器试图在火星表面着陆时失踪。一个故障评估委员会调查了故障，认定出现故障的原因极可能是一个数据位被意外置位。最令人震惊的问题是为什么没有在内部测试时发现呢。

从理论上看，着陆的计划是这样的：当探测器向火星表面降落时，它将打开降落伞减缓探测器的下降速度。降落伞打开几秒钟后，探测器的三条腿将迅速撑开，并锁定位置，准备着陆。当探测器离地面 1800 米时，它将丢弃降落伞，点燃着陆推进器，缓缓地降落到地面。

美国航天局为了省钱，简化了确定何时关闭着陆推进器的装置。为了替代其他太空船上使用的贵重雷达，他们在探测器的脚部装了一个廉价的触点开关，在计算机中设置一个数据位来控制触点开关关闭燃料。很简单，探测器的发动机需要一直点火工作，直到脚"着地"为止。

遗憾的是，故障评估委员会在测试中发现，许多情况下，当探测器的脚迅速撑开准备着陆时，机械震动也会触发着陆触点开关，设置致命的错误数据位。设想探测器开始着陆时，计算机极有可能关闭着陆推进器，这样火星极地登陆者号探测器飞船下坠 1800 米之后冲向地面，撞成碎片。

结果是灾难性的，但背后的原因却很简单。登陆探测器经过了多个小组测试。其中一个小组测试飞船的脚折叠过程，另一个小组测试此后的着陆过程。前一个小组不去注意着陆数据是否置位——这不是他们负责的范围；后一个小组总是在开始复位之前复位计算机，清除数据位。双方独立工作都做得很好，但合在一起就不是这样了。

(5) 金山词霸缺陷

在国内，"金山词霸"是一个很著名的词典软件，应用范围极大，对使用中文操作的用户帮助很大，但它也存在不少缺陷。例如输入"cube"，词霸会在示例中显示 $3^3=9$ 的错误；又如，如果用鼠标取词"dynamically"(力学，动力学)，词霸会出现其他不同的单词"dynamite *n.* 炸药"的显示错误。

(6) 英特尔奔腾浮点除法缺陷

在计算机的"计算器"程序中输入以下算式：

(4195835/3145727)*3145727－4195835

如果答案是0,就说明计算机没问题。如果得出别的结果,就表示计算机使用的是带有浮点除法软件缺陷的老式英特尔奔腾处理器——这个软件缺陷被烧录在一个计算机芯片中,并在制作过程中反复生产。

1994年10月30日,弗吉尼亚州Lynchburg学院的Thomas R. Nicely博士在他的一个实验中,用奔腾PC机解决一个除法问题时,记录了一个想不到的结果,得出了错误的结论。他把发现的问题放到因特网上,随后引发了一场风暴,成千上万的人发现了同样的问题,并且发现在另外一些情形下也会得出错误的结果。万幸的是,这种情况很少见,仅仅在进行精度要求很高的数学、科学和工程计算中才会导致错误。大多数用来进行税务处理和商务应用的用户根本不会遇到此类问题。

这件事情引人关注的并不是这个软件缺陷,而是英特尔公司解决问题的方式:

- 他们的软件测试工程师在芯片发布之前进行内部测试时已经发现了这个问题。英特尔的管理层认为这没有严重到要保证修正,甚至公开的程度。
- 当软件缺陷被发现时,英特尔通过新闻发布和公开声明试图弱化这个问题的已知严重性。
- 受到压力时,英特尔承诺更换有问题的芯片,但要求用户必须证明自己受到缺陷的影响。

舆论哗然。互联网新闻组里充斥着愤怒的客户要求英特尔解决问题的呼声。新闻报道把英特尔公司描绘成不关心客户和缺乏诚信者。最后,英特尔为自己处理软件缺陷的行为道歉并拿出4亿多美元来支付更换问题芯片的费用。现在英特尔在Web站点上报告已发现的问题,并认真查看客户在互联网新闻组里所留下的反馈意见。

2. 软件缺陷的定义

从上述的案例中可以看到软件发生错误时将造成灾难性危害或对用户产生各种影响。

在这些事件中,显然软件未按预期目标运行。作为软件测试员,可能会发现大多数缺陷不如上面所列举的实例那么明显,而对于一些简单而细微的错误,很难做到真正区分哪些是真正的错误,哪些不是。对于软件存在的各种问题我们都称为软件缺陷或软件故障。在英文中人们喜欢用一个不贴切但已经专用的词“bug”表示。

软件缺陷即计算机系统或者程序中存在的任何一种破坏正常运行能力的问题、错误,或者隐藏的功能缺陷、瑕疵。缺陷会导致软件产品在某种程度上不能满足用户的需要。对于软件缺陷的准确定义,通常有以下5条描述:

① 软件未实现产品说明书要求的功能。

② 软件出现了产品说明书指明不会出现的错误。

③ 软件实现了产品说明书未提到的功能。

④ 软件实现了产品说明书虽未明确指出但应该实现的目标。

⑤ 软件难以理解，不易使用，运行缓慢或者终端用户认为不好。

为了更好地理解每一条规则，我们以计算器为例进行说明。

计算器的产品说明书声称它能够准确无误地进行加、减、乘、除运算。当你拿到计算器后，按下（＋）键，结果什么反应也没有，根据第①条规则，这是一个缺陷。假如得到错误答案，根据第①条规则，这同样是一个缺陷。

若产品说明书声称计算器永远不会崩溃、锁死或者停止反应。当你任意敲键盘，计算器停止接受输入，根据第②条规则，这是一个缺陷。

若用计算器进行测试，发现除了加、减、乘、除之外它还可以求平方根，说明书中从没提到这一功能，根据第③条规则，这是软件缺陷。软件实现了产品说明书未提到的功能。

若在测试计算器时，发现电池没电会导致计算不正确，但产品说明书未指出这个问题。根据第④条规则，这是个缺陷。

第⑤条规则是全面的。如果软件测试员发现某些地方不对劲，无论什么原因，都要认定为缺陷。如“＝”键布置的位置极其不好按；或在明亮光下显示屏难以看清。根据第⑤条规则，这些都是缺陷。

美国商务部国家标准和技术研究所（NIST）进行的一项研究表明，软件中的 bug 每年给美国经济造成的损失高达 595 亿美元。说明软件中存在的缺陷所造成的损失是巨大的，从反面又一次证明软件测试的重要性。如何尽早彻底地发现软件中存在的缺陷是一项非常复杂，需要创造性和高度智慧的工作。同时，软件的缺陷是软件开发过程中的重要属性，反映软件开发过程中需求分析、功能设计、用户界面设计、编程等环节所隐含的问题，也为项目管理、过程改造提供了许多信息。

3. 软件缺陷的原因

软件缺陷的产生，首先是不可避免的。其次，我们可以从软件本身，团队工作和技术问题等多个方面分析，将比较容易确定造成软件缺陷的原因归纳如下。

（1）技术问题

- 算法错误。
- 语法错误。
- 计算和精度问题。
- 系统结构不合理，造成系统性能问题。
- 接口参数不匹配出现问题。

（2）团队工作

- 系统分析时对客户的需求不是十分清楚，或者和用户的沟通存在一些困难。
- 不同阶段的开发人员相互理解不一致，软件设计对需求分析结果的理解偏差，编

程人员对系统设计规格说明书中某些内容重视不够，或存在着误解。

- 设计或编程上的一些假定或依赖性，没有得到充分地沟通。

(3) 软件本身

- 文档错误、内容不正确或拼写错误。
- 数据考虑不周全引起强度或负载问题。
- 对边界考虑不够周全，漏掉某几个边界条件造成的错误。
- 对一些实时应用系统，保证精确的时间同步，否则容易引起时间上不协调、不一致性带来的问题。
- 没有考虑系统崩溃后在系统安全性、可靠性的隐患。
- 硬件或系统软件上存在的错误。
- 软件开发标准或过程上的错误。

4. 软件缺陷的组成

我们知道软件缺陷是由很多原因造成的，如果把它们按需求分析结果——规格说明书、系统设计结果、编程的代码等归类起来，比较后发现，规格说明书是软件缺陷出现最多的地方，如图 1-1 所示。

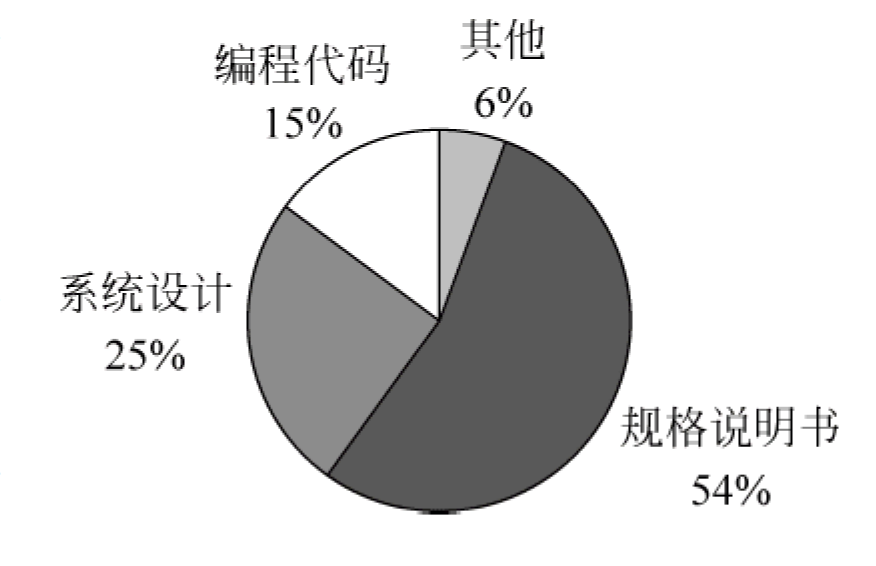


图 1-1 软件缺陷构成示意图

软件产品规格说明书为什么是软件缺陷存在最多的地方，主要原因有以下几种。

(1) 用户一般是非计算机专业人员，软件开发人员和用户的沟通存在较大困难，对要开发的产品功能理解不一致。

(2) 由于软件产品还没有设计、开发，完全靠想象去描述系统的实现结果，所以有些特性还不够清晰。

(3) 需求变化的不一致性。用户的需求总是在不断变化的，这些变化如果没有在产品规格说明书中得到正确的描述，容易引起前后文，上下文的矛盾。

(4) 对规格说明书不够重视，在规格说明书的设计和写作上投入的人力、时间不足。

(5) 没有在整个开发队伍中进行充分沟通，有时只有设计师或项目经理得到比较多的信息。

5. 软件缺陷的修复费用

软件不仅仅是表面上的那些东西——通常要靠有计划、有条理的开发过程来实现。从开始到计划、编程、测试，到公开使用的过程中，都有可能发现软件缺陷。

软件缺陷造成的修复费用呈指数级地增长——也就是说，随着时间的推移，由于软件缺陷造成的修复费用呈十倍地增长。当早期编写产品说明书时发现并修复缺陷，费用只

要 1 美元甚至更少。同样的缺陷如果直到软件编写完成开始测试时才发现，费用可能要 10～100 美元。如果是客户发现的，费用可能达到数千甚至数百万美元。

举一个例子来说明，比如前面的迪斯尼狮子王实例。问题的根本原因是软件无法在流行的 PC 平台上运行。假如早在编写产品说明书时，有人已经研究过什么 PC 机流行，并且明确指出软件需要在该种配置上设计和测试，付出的代价小得几乎可以忽略不计。如果没有这样做，还有一个补救措施是，软件测试员去搜集流行 PC 样机并在其上验证。他们可能会发现软件缺陷，但是修复费用要高得多，因为软件必须调试、修改、再测试。开发小组还应当把软件的初期版本分发给一小部分客户进行试用 这叫 Beta 测试。那些被挑选出来代表庞大市场的客户可能会发现问题。然而实际的情况是，缺陷被完全忽视，直到成千上万的光盘被压制和销售出去。而迪斯尼公司最终支付了客户投诉电话费、产品召回、更换光盘，以及又一轮调试、修改和测试的费用。如果严重的软件缺陷到了客户那里，就足以耗尽整个产品的利润。

1.1.2　软件测试技术的发展历史和现状

1. 软件测试技术的发展历史

随着计算机的诞生——在软件行业发展初期就已经开始实施软件测试，但这一阶段还没有系统意义上的软件测试，更多的是一种类似调试的测试。测试是没有计划和方法的，测试用例的设计和选取也都是根据测试人员的经验随机进行的，大多数测试的目的是为了证明系统可以正常运行。

20 世纪 50 年代后期到 20 世纪 60 年代，随着计算机软件的发展，用于计算机编程的各种高级语言也相继诞生，测试的重点也逐步转入到使用高级语言编写的软件系统中来，但程序的复杂性远远超过了以前。尽管如此，由于受到硬件的制约，在计算机系统中，软件仍然处于次要位置。软件正确性的把握仍然主要依赖于编程人员的技术水平。因此，这一时期软件测试的理论和方法发展比较缓慢。

20 世纪 70 年代以后，随着计算机处理速度的提高，存储器容量的快速增加，软件在整个计算机系统中的地位变得越来越重要。随着软件开发技术的成熟和完善，软件的规模也越来越大，复杂度也大大增加。因此，软件的可靠性面临着前所未有的危机，给软件测试工作带来了更大的挑战，很多测试理论和测试方法应运而生，逐渐形成了一套完整的体系，培养和造就了一批批出色的软件测试人才。

如今在软件产业化发展的大趋势下，人们对软件质量、成本和进度的要求也越来越高，软件质量的控制已经不仅仅是传统意义上的软件测试。传统软件的测试大多是基于代码运行的，并且常常是软件开发的后期才开始进行，但大量研究表明，设计活动引入的错误占软件开发过程中出现的所有错误数量的 50%～65%。因此，越来越多的声音呼吁，要求有一个规范的软件开发过程。而在整个软件开发过程中，测试已经不再只是基于

程序代码进行的活动,而是一个基于整个软件生命周期的质量控制活动,贯穿于软件开发的各个阶段。

2. 软件测试的现状

在我国,软件测试可能算不上一个真正的产业,软件开发企业对软件测试认识淡薄,软件测试人员与软件开发人员往往比例失调,而在发达国家和地区软件测试已经成为了一个产业,微软的开发工程师与测试工程师的比例是 1∶2,国内一般公司是 6∶1。很多人认为导致这种现状产生的原因是与我们接受的传统教育和开发习惯有相当大的关系。软件行业相对于其他一些行业来说是相当年轻的,开发过程包含了需求分析、设计、编程、测试和维护等工作,由于软件业的历史年轻,而且一般人认为,开发周期前面的工作没有完善之前,比较难于考虑到后面的工作。因此,我们可以看到软件工作大部分的精力都投入在了需求分析、设计、编程 3 个阶段的开发,造成了这些方面方法论的快速发展,而忽视了测试工作。

总之,与一些发达国家相比,国内软件测试工作还存在一定的差距。主要体现在软件测试意识以及测试理论的研究,大型测试工具软件的开发以及从业人员数量等方面。其实,这与中国整体软件的发展水平是一致的,因为我国整体的软件产业水平和软件发达国家的水平相比有较大的差距,而作为软件产业重要一环的软件测试,必然也存在着不小的差距。但是,我们在软件测试实现方面并不比国外差,国际上优秀的测试工具,我们基本都有,这些工具所体现的思想我们也有深刻的理解,很多大型系统在国内都得到了很好的测试。

1.2 软件测试的基本理论

软件测试在软件生命周期中横跨两个阶段。通常在编写出每个模块之后就对它进行必要的测试(称为单元测试),模块的编写者和测试者是同一个人,编码和单元测试属于软件生命周期的同一个阶段。在结束这个阶段之后,对软件系统还要进行各种综合测试,这是软件生命周期中另一个独立的阶段,通常由专门的测试人员来完成这项工作。

目前,人们越来越重视软件测试,软件测试的工作量往往占到软件开发总工作量的 40%以上。在特殊情况下,测试那些重大的软件,例如核反应堆监控软件,其测试费用可能相当于软件工程其他步骤总成本的三倍到五倍。因此,我们必须高度重视软件测试工作,绝不能认为写出程序代码之后软件开发工作就完成了。

1.2.1 软件测试的定义和目标

1. 软件测试的定义

人们对于软件测试的目的可能会存在着这样的认识:软件测试是为了证明程序是正

确的。实际上,这种认识是错误的。因为如果表明程序是正确的而进行测试,就会设计一些不易暴露错误的测试方案,也不会主动去检测、排除程序中可能存在的一些隐患。显然,这样的测试对于发现程序中的错误,完善和提高软件的质量作用不大。因为程序在实际运行中会遇到各种各样的实际问题,而这些问题可能是我们在设计时没有考虑到的,所以在设计测试方案时,就应该尽量让它能发现程序中的错误,从而在软件投入运行之前就将这些错误改正,最终把一个高质量的软件系统交给用户使用。

通常对软件测试的定义有如下描述:软件测试是为了发现程序中的错误而执行程序的过程。具体说,它是根据软件开发各阶段的规格说明和程序的内部结构而精心设计出一批测试用例,并利用测试用例来运行程序,以发现程序错误的过程。

正确认识测试的目的是十分必要的,只有这样,才能设计出最能暴露错误的测试方案。此外,应该认识到:测试只能证明程序中错误的存在,但不能证明程序中没有错误。因为即使经过了最严格的测试之后,仍然可能还有没被发现的错误存在于程序中,所以说测试只能查出程序中的错误,但不能证明程序没有错误。

2. 软件测试的目标

软件测试工作是非常必要的,测试的目的就在于在软件投入运行之前,尽可能多地发现软件中的错误。软件测试是对软件规格说明、设计和编码的最后复审,是软件质量保证的关键步骤。

实现这个目的关键是如何合理地设计测试用例,在设计测试用例时,要着重考虑那些易于发现程序错误的方法策略与具体数据。

综上所述,软件测试的目的包括以下三点:

(1) 测试是程序的执行过程,目的在于发现错误,不能证明程序的正确性,仅限于处理有限种的情况。

(2) 检查系统是否满足需求,这也是测试的期望目标。

(3) 一个好的测试用例在于发现还未曾发现的错误;成功的测试是发现了错误的测试。

1.2.2 软件测试的标准

软件测试的标准是站在用户的角度,对产品进行全面测试,尽早、尽可能多地发现缺陷(bug),并负责跟踪和分析产品的问题,对不足之处提出质疑和改进意见。

软件测试标准如下:

(1) 软件测试的目标在于揭示错误。测试人员要始终站在用户的角度去看问题,系统中最严重的错误的是那些导致程序无法满足用户需求的错误。

(2) 软件测试必须基于"质量第一"的思想去开展各项工作。

(3) 事先定义好产品的质量标准。只有建立了质量标准,才能根据测试的结果,对产

品的质量进行分析和评估。

(4) 软件项目一启动,软件测试也就开始了,而不是等程序写完,才开始进行测试。

(5) 测试用例是设计出来的,不是写出来的,所以要根据测试的目的,采用相应的方法去设计测试用例,从而提高测试的效率,更多的发现错误,提高程序的可靠性。

(6) 对发现错误较多的程序段,应进行更深入的测试。

1.2.3 软件测试的原则

各种统计数据显示,软件开发过程中发现缺陷的时间越晚,修复它所花费的成本就越大,因此在需求分析阶段就应当有测试的介入。因为软件测试的对象不仅仅是程序编码,应当对软件开发过程中产生的所有产品都进行测试。这就像造桥梁一样,在图纸上面设计好桥梁的结构之后,我们只有对图纸进行仔细地审查后,才能进行施工。

人们普遍存在着一种观念,认为可以对程序进行完全的测试。如:

- 许多管理者认为存在完全测试的可能性,因此要求员工这样做,并在彼此间确认正在这样做。
- 某些软件测试公司在产品销售说明中保证他们能对软件进行完全的测试。
- 有时,测试覆盖率分析人员为了推销自己,也宣称自己能够分析是否已经对代码进行了完全测试;或者能够指出下一步还需要做什么测试就能够进行完全的测试。
- 许多销售人员向客户强调他们的软件产品经过了完全的测试,彻底没有错误。
- 一些测试人员也相信存在着完全测试的秘诀,甚至为实现这种想法而吃尽了苦头,忍受了数次失败和挫折。但实际上,无论工作得多么辛苦,计划得多么周密,投入的时间多长,人力和物力资源多大,仍然无法做到充分的测试,仍然会遗漏缺陷。

对一个程序进行完全测试就意味着在测试结束之后,再也不会发现其他的软件错误了。其实,这是不可能的,充其量是测试人员的一种美好的愿望而已。

除了测试人员之外,程序员在编写完每段编码之后,或者在每个子模块完成后,都要进行认真测试,这样就可以在最早的时间发现一些潜在的问题并加以解决。之所以这样做,是由于在测试过程中我们要避免一些人为的和主观因素的干扰。我们知道,开发和测试是互为相反的行为过程,两者有着本质的不同。在程序员完成大量的设计和编码之后,让他否定自己所做的工作,是非常不易的,可以说很少有人能有这样的心态。另外一个原因就是系统需求的错误不易被发现,如果程序员检查自己的代码,那么他对系统需求的理解缺乏客观性,往往存在着对问题叙述或说明的误解,不难想象带有错误认识的程序员是很难发现自己程序存在的问题的。

软件测试的本质就是针对要测试的内容确定一组测试用例。测试用例至少应当包括如下几个基本信息:

• 在执行测试用例之前，应满足的前提条件；
• 输入的数据（合理的、不合理的）；
• 预期的输出（包括后果和实际输出）。

有经验的测试人员会发现，在进行软件测试的过程中，常发生软件缺陷“扎堆”的现象，因此当我们在某一部分发现了很多错误时，应当进一步仔细测试是否还包含了更多的软件缺陷。

软件缺陷“扎堆”的现象常见形式有：

• 对话框的某个控件功能不起作用，可能其他控件的功能也不起作用。
• 某个文本框不能正确显示双字节字符，则其他文本框也可能不支持双字节字符。
• 联机帮助某段文字的翻译包含了很多错误，与其相邻的上下段的文字可能也包含很多的语言质量问题。
• 安装文件某个对话框的“上一步”或“下一步”按钮被截断，则这两个按钮在其他对话框中也可能被截断。

良好的开始是成功的一半。合理的测试计划有助于测试工作顺利有序地进行，因此要求在对软件进行测试之前所做的测试计划中，应该结合多种针对性强的测试方法，列出所有可使用资源，建立一个正确的测试目标，本着严谨、准确的原则，周密细致地做好测试前期的准备工作，避免测试的随意性。尤其是要尽量科学合理地安排测试时间，并留出一定的机动时间，防止意外情况的发生，以免出现测试时间不够用，甚至使很多测试工作不能正常进行的情况，尽量降低测试风险。

软件测试的目标是想以最少的时间和人力找出软件中潜在的各种错误和缺陷。如果成功地实施了测试，就能够发现软件中的错误。

根据这样的测试目的，软件测试的原则应该是：

（1）应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭。坚持在软件开发的各个阶段的技术评审，这样才能在开发过程中尽早发现和预防错误，把出现的错误克服在早期，杜绝某些隐患，提高软件质量。

（2）测试用例应由测试输入数据和与之对应的预期输出结果这两部分组成。如果对测试输入数据没有给出预期的程序输出结果，那么就缺少了检验实测结果的基准，就有可能把一个似是而非的错误结果当成正确结果。

（3）程序员应避免检查自己的程序。如果由别人来测试程序员编写的程序，可能会更客观、更有效、更容易取得成功。

（4）在设计测试用例时，应当包括合理的输入条件和不合理的输入条件。

合理的输入条件是指能验证程序正确的输入条件，而不合理的输入条件是指异常的、临界的、可能引起问题变异的输入条件。因此，软件系统处理非法命令的能力也必须在测试时受到检验。用不合理的输入条件测试程序时，往往比用合理的输入条件进行测试能

发现更多的错误。

(5) 充分注意测试中的群集现象。测试时不要以为找到了几个错误问题就已解决，不需要继续测试了。应当对错误群集的程序段进行重点测试，以提高测试投资的效益。

(6) 严格执行测试计划，排除测试的随意性。对于测试计划，要明确规定，不要随意解释。

(7) 应当对每一个测试结果做全面检查。这是一条最明显的原则，但常常被忽视。必须对预期的输出结果明确定义，对实测的结果仔细分析检查，抓住关键，暴露错误。

(8) 妥善保存测试计划、测试用例、出错统计和最终分析报告，为维护提供方便。

1.2.4 软件测试的分类

从不同的角度，可以把软件测试技术分成不同种类。

1. 从是否需要执行被测软件的角度分类

从是否需要执行被测软件的角度，软件测试可分为静态测试(Static Testing)和动态测试(Dynamic Testing)。顾名思义，静态测试就是通过对被测程序的静态审查，发现代码中潜在的错误。它一般用人工方式脱机完成，故亦称人工测试或代码评审(Code Review)；也可借助于静态分析器在机器上以自动方式进行检查，但不要求程序本身在机器上运行。按照评审的不同组织形式，代码评审又可分为代码会审、走查、办公桌检查、同行评分 4 种。对某个具体的程序，通常只使用一种评审方式。

动态测试的对象必须是能够由计算机真正运行的被测试的程序。它分为黑盒测试和白盒测试，在第 3 章、第 4 章中将会详细介绍。

2. 从软件测试用例设计方法的角度分类

从软件测试用例设计方法的角度，可分为黑盒测试(Black-Box Testing)和白盒测试(White-Box Testing)。

黑盒测试是一种从用户角度出发的测试，又称为功能测试、数据驱动测试和基于规格说明的测试。使用这种方法进行测试时，把被测试程序当作一个黑盒，忽略程序内部的结构的特性，测试者在只知道该程序输入和输出之间的关系或程序功能的情况下，依靠能够反映这一关系和程序功能需求规格的说明书，来确定测试用例和推断测试结果的正确性。简单地说，若测试用例的设计是基于产品的功能，目的是检查程序各个功能是否实现，并检查其中的功能错误，则这种测试方法称为黑盒。

白盒测试基于产品的内部结构来进行测试，检查内部操作是否按规定执行，软件各个部分功能是否得到充分利用。白盒测试又称为结构测试、逻辑驱动测试或基于程序的测试。即根据被测程序的内部结构设计测试用例，测试者需要事先了解被测试程序的结构。

3. 从软件测试的策略和过程的角度分类

按照软件测试的策略和过程分类，软件测试可分为单元测试(Unit Testing)、集成测

试(Integration Testing)、确认测试(Validation Testing)、系统测试(System Testing)和验收测试(Verification Testing)。

单元测试是针对每个单元的测试,是软件测试的最小单位。它确保每个模块能正常工作。单元测试多数使用白盒测试,用以发现内部错误。

集成测试是对已测试过的模块进行组装,进行集成测试的目的主要在于检验与软件设计相关的程序结构问题。集成测试一般通过黑盒测试方法来完成。

确认测试是检验所开发的软件能否满足所有功能和性能需求的最后手段,通常采用黑盒测试方法。

系统测试的主要任务是检测被测软件与系统的其他部分的协调性。

验收测试是软件产品质量的最后一关。这一环节,测试主要从用户的角度着手,其参与者主要是用户和少量的程序开发人员。

1.3　软件测试与软件开发

1. 软件测试与软件开发各阶段的关系

软件开发过程是一个自顶向下、逐步细化的过程,首先在软件计划阶段定义了软件的作用域,然后进行软件需求分析,建立软件的数据域、功能和性能需求、约束和一些有效性准则。接着进入软件开发,首先是软件设计,然后再把设计用某种程序设计语言转换成程序代码。而测试过程则是依相反的顺序安排的自底向上、逐步集成的过程,低一级测试为上一级测试准备条件。此外还有两者平行地进行测试。

如图 1-2 所示是软件测试与软件开发过程的关系图。首先对每一个程序模块进行单元测试,消除程序模块内部在逻辑上和功能上的错误和缺陷。再对照软件设计进行集成测试,检测和排除子系统(或系统)结构上的错误。随后再对照需求,进行确认测试。最后从系统全体出发,运行系统,看是否满足要求。

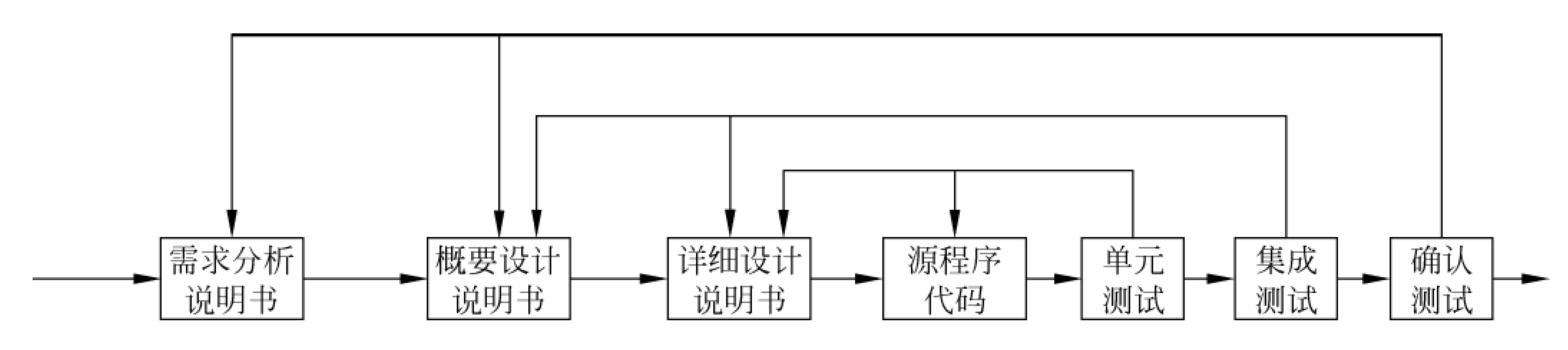


图 1-2　软件测试与软件开发过程的关系

2. 测试与开发的并行性

在软件的需求得到确认并通过评审后,概要设计工作和测试计划制定设计工作就要并行进行。如果系统模块已经建立,对各个模块的详细设计、编码、单元测试等工作又可

并行。待每个模块完成后，可以进行集成测试、系统测试。软件测试与软件开发并行流程如图 1-3 所示。

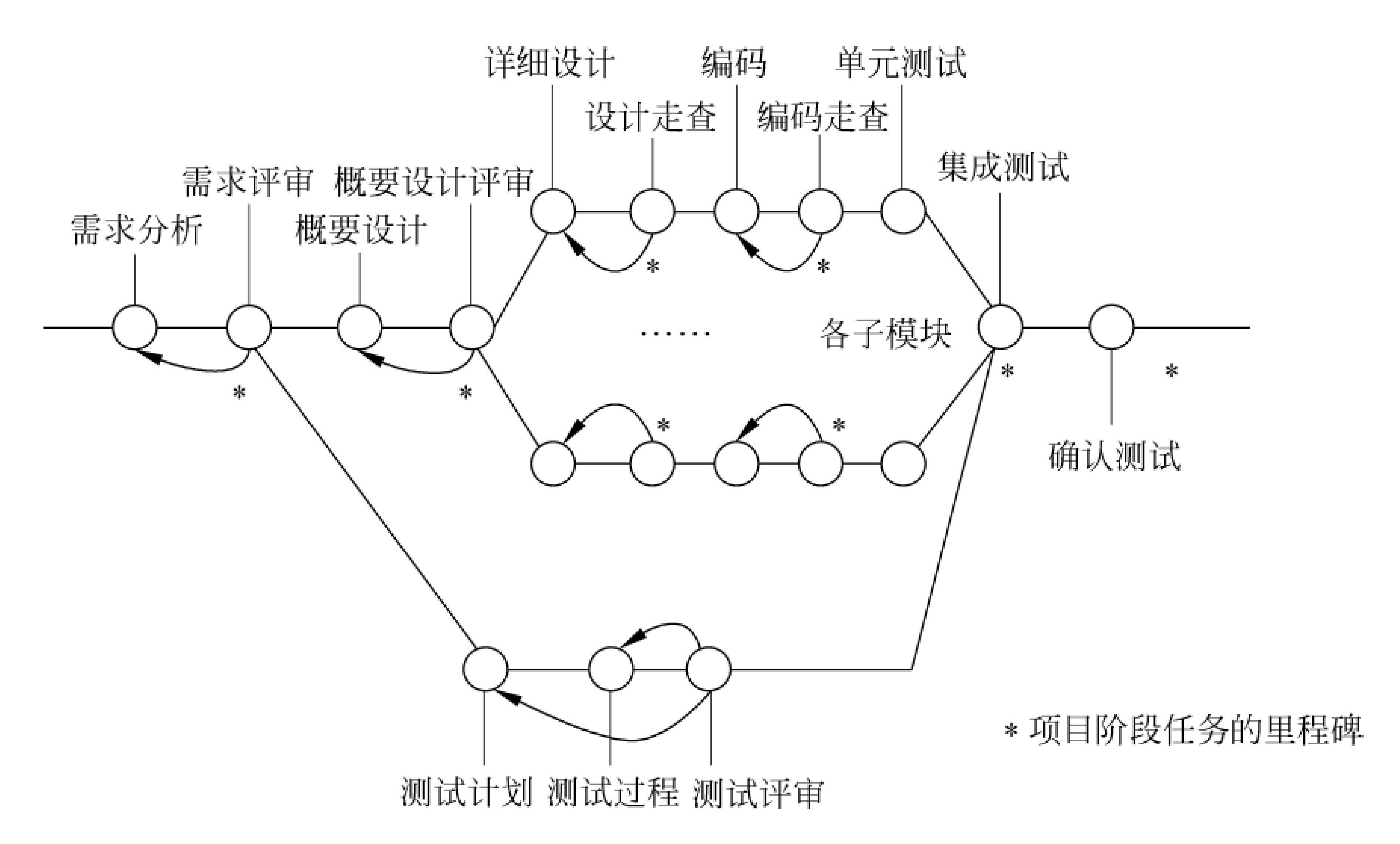


图 1-3　软件测试与软件开发的并行性

3. 测试与开发模型

软件测试不仅仅是执行测试，而且是一个包含很多复杂活动的过程，并且这些过程应该贯穿于整个软件的开发过程。在软件开发过程中，应该什么时候进行测试，如何更好地把软件开发和测试活动集成到一起？其实这也是软件测试工作人员必须考虑的问题，因为只有这样，才能提高软件测试工作的效率，提高软件产品的质量，最大限度地降低软件开发与测试的成本，减少重复劳动。如图 1-4 所示，即为软件测试与开发的完整流程。

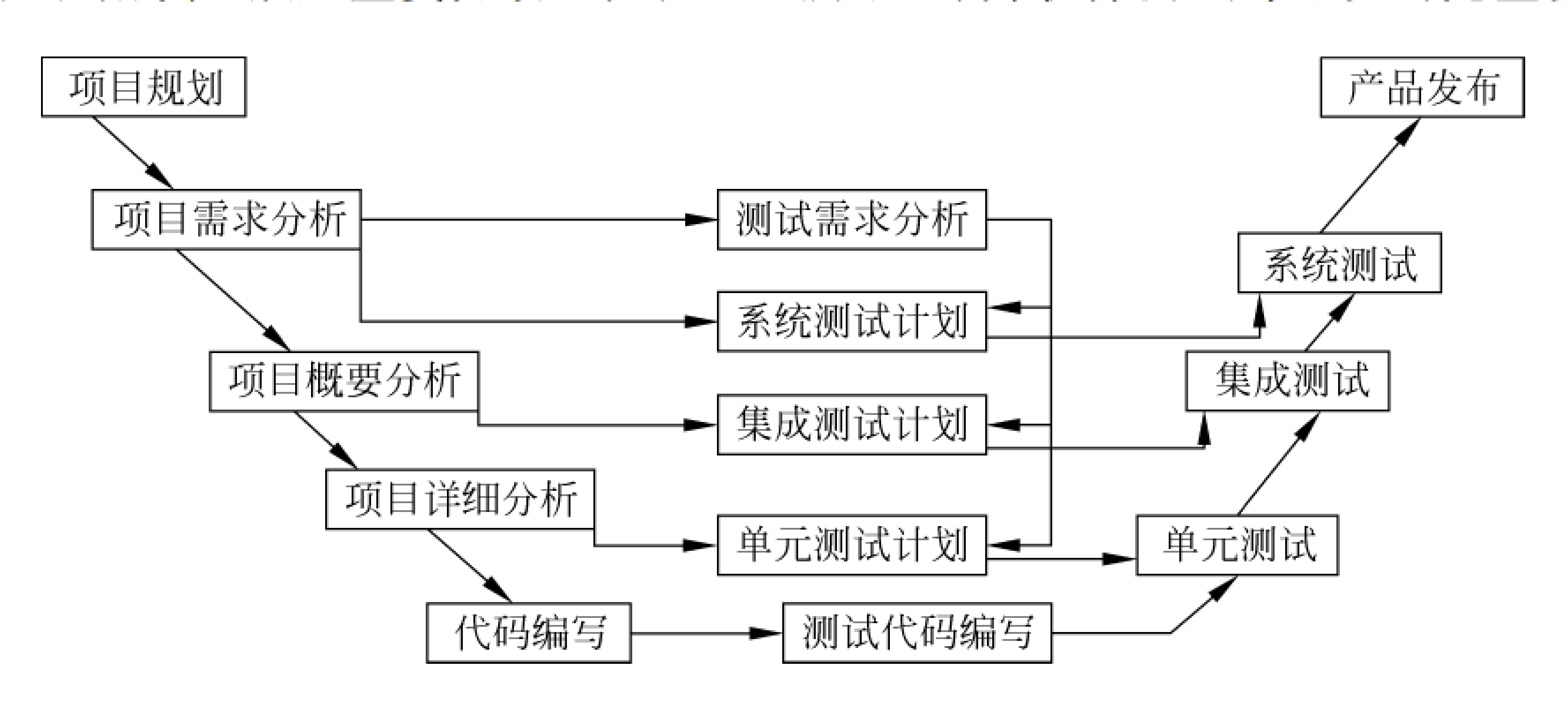


图 1-4　完整的软件开发和测试流程

小　　结

本章从软件缺陷实例为出发点介绍了软件测试背景和测试发展的历程，以及它在国内的发展状况。随着软件开发过程和开发技术的不断改进，软件测试理论和方法也在不断完善，测试工具也在蓬勃发展。软件测试是软件质量保证的手段，本章讲述了软件测试的定义，明确了软件测试的目标就是以最少的时间和人力找出软件中潜在的各种错误和缺陷，阐述了软件测试执行的标准和软件测试的原则。从不同角度，对软件测试进行了分类：从是否需要执行被测软件的角度可分为静态测试和动态测试；从软件测试用例设计方法的角度可分为黑盒测试和白盒测试；从软件测试的策略和过程的角度又可分为单元测试、集成测试、确认测试、系统测试、验收测试。最后介绍了软件开发与软件测试的相辅相成的关系。

习　　题

1. 名词解释：

软件缺陷、软件测试、静态测试、动态测试、黑盒测试、白盒测试、单元测试、集成测试。

2. 简述软件测试发展的历史及软件测试的现状。

3. 谈谈你对软件测试的重要性的理解。

4. 简述软件测试的目标及标准。

5. 简述软件测试的原则。

6. 简述软件测试与软件开发的关系。

第 2 章

软件测试过程与策略

本章概述

软件产品种类繁多，测试过程千变万化，为了能够找到系统中绝大部分的软件缺陷，必须构建各种行之有效的测试方法与策略。

本章通过详细分析，介绍了软件测试的复杂性和经济性；通过讲述软件测试的整个流程，从而了解单元测试、集成测试、确认测试、系统测试和验收测试等基本测试方法；通过比较分析，介绍了静态与动态测试、黑盒与白盒测试的基本策略。

2.1　软件测试的复杂性与经济性分析

人们在对软件工程开发的常规认识中，认为开发程序是一个复杂而困难的过程，需要花费大量的人力、物力和时间，而测试一个程序则比较容易，不需要花费太多的精力。这其实是人们对软件工程开发过程理解上的一个误区。在实际的软件开发过程中，作为现代软件开发工业一个非常重要的组成部分，软件测试正扮演着越来越重要的角色。随着软件规模的不断扩大，如何在有限的条件下对被开发软件进行有效的测试正成为软件工程中一个非常关键的课题。

2.1.1　软件测试的复杂性

设计测试用例是一项细致并且需要具备高度技巧的工作，稍有不慎就会顾此失彼，发生不应有的疏漏。下面分析了容易出现问题的根源。

(1) 完全测试是不现实的

在实际的软件测试工作中，由于软件测试情况数量极其巨大，不论采用什么方法，都不可能进行完全彻底的测试。所谓彻底测试，就是让被测程序在一切可能的输入情况下全部执行一遍。通常也称这种测试为“穷举测试”。

穷举测试会引起以下几种问题：

- 输入量太大；
- 输出结果太多；
- 软件执行路径太多；
- 说明书存在主观性。

由于以上问题的存在，使得在大多数的软件测试过程中，穷举测试几乎是不可能的。在软件的使用过程中，人们不仅要进行合法的输入，若出现某些意外情况，可能还要发生种种不合法的输入。这样的测试情况可能出现无穷多个，所以测试人员既要测试所有合法的输入，还要对那些不合法但是可能的输入进行测试。

例如，对于常用的画图板程序，如果测试人员受命于使用穷举测试来进行，那么首先要对直线的画图进行测试，把直线中最小的两个相邻点，一个一个地延长直至最大的点；然后是考虑反方向的画图，再是斜线的画图。将所有可能的直线画图全部测试完成之后，还要考虑其他图像的各种画法，一个一个地在理论上将所有可能发生的情况全部测试完毕后，再将可能出现的不同图形的叠加全部实现，当然这里还不包括色彩的运用。按照上述思路一个一个的测试，单是合法输入就接近无穷多个，使得在理论上根本无法进行穷举测试。在实际的使用过程中，测试人员还要考虑到包括随机出现的各种突发情况，比如用户不小心撞到键盘引起某个误操作。经典著作《软件测试技巧》的作者 G. J. Myers 在 1979 年描述了一个只包含 loop 循环和 if 语句的简单程序。可以使用不同的语言将其写成 20 行左右的代码，但是这样简短的语句却有着十万亿条路径。面对这样一个庞大的数字，即便是一个有经验的优秀的软件测试员也需要十亿年才能完成全部测试，而且在实际应用中，此类程序是非常有可能出现的。

E. W. Dijkstra 的一句名言对测试的不彻底性作了很好的注解：“程序测试只能证明错误的存在，但不能证明错误的不存在”。由于穷举测试工作量太大，实践上行不通，这就注定了一切实际测试都是不彻底的，也就不能够保证被测试程序在理论上不存在遗留的错误。

（2）软件测试是有风险的

穷举测试的不可行性使得大多数软件在进行测试的时候只能采取非穷举测试，这又意味着一种冒险。比如在使用 Microsoft Office 工具中的 Word 时，可以作这样的一个测试：①新建一个 Word 文档；②在文档中输入汉字“胡”；③设置其字体属性为“隶书”，字号为初号，效果为“空心”；④将页面的显示比例设为“500％”。这时在“胡”字的内部会出现“胡万进印”4 个字。类似问题在实际测试中如果不使用穷举测试是很难发现的，而如果在软件投入市场时才发现则修复代价就会非常高。这就会产生一个矛盾：软件测试员不能做到完全的测试，不完全测试又不能证明软件的百分之百的可靠。那么如何在这两者的矛盾中找到一个相对的平衡点呢？

从如图 2-1 所示的最优测试量示意图可以观察到，当软件缺陷降低到某一数值后，随着测试量的不断上升软件缺陷并没有明显地下降。这是软件测试工作中需要注意的重要问题。如何把测试数据量巨大的软件测试减少到可以控制的范围，如何针对风险做出最明智的选择是软件测试人员必须把握的关键问题。

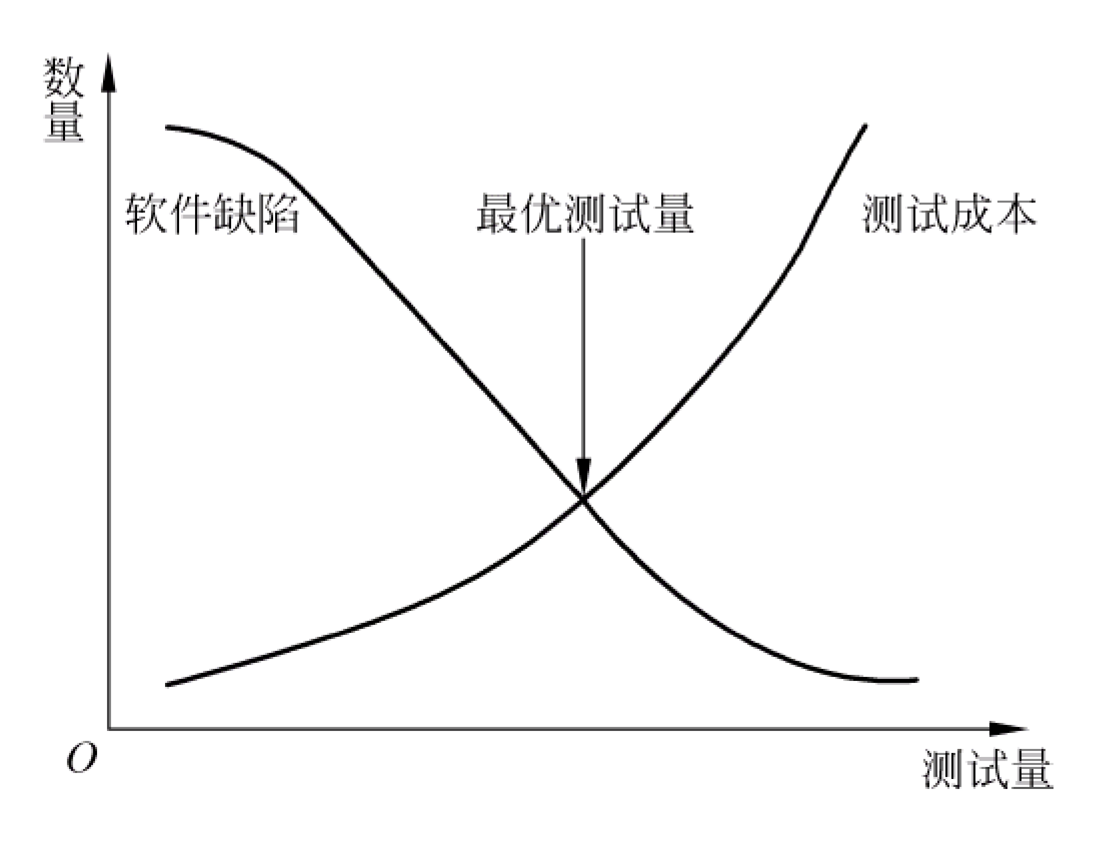


图 2-1　最优测试量示意图

图 2-1 的最优测试量示意图说明了发现软件缺陷数量和测试量之间的关系，随着测试量的增加，测试成本将呈几何数级上升，而软件缺陷降低到某一数值之后将没有明显的变化，最优测量值就是这两条曲线的交点。

对于软件测试数据量巨大的问题没有十全十美的解决办法，采取最优测试量只是在两者中的一种妥协。然而糟糕的是矛盾还不止于此，测试量会随着时间的推移而发生改变。在当今竞争激烈的市场里，争取时间可能是制胜的关键，这本身就使软件的开发与测试出现矛盾。使情况更加复杂的是，当一种新的技术或者新的标准出现时，可能人们会对软件是否十全十美并不在意了。在这种情况下，要进行多长时间的测试就更是一个值得商榷的问题。

微软公司在研制 Windows 操作系统的第一个版本时已经落后于对手了。但是为了能抢得第一部应用于 PC 机的图形界面操作系统这一市场先机，微软公司一面大做广告宣传，一面在公司里加班加点。为了能尽快进行产品的发布，在没有进行可靠的测试验证的情况下，微软公司就公布了自己的 Windows 操作系统。虽然这时的 Windows 操作系统漏洞百出，但是仍然赢得了绝大部分的市场份额。反观微软公司的竞争对手，虽然对产品进行了完善的测试与验证，但这时候已经没有人来关注他们的产品了，大家都在兴致勃勃地讨论着 Windows 操作系统，竞争公司最终退出了这一市场。

当然，上面的例子只是在市场初期的特殊现象。如果市场分配格局已经建立起来，那么就应该针对合适的目标加大测试量，提高产品质量。但是在不同的市场时期如何决定测试量的多少对于一个软件开发公司仍然是一个十分重要的课题，这不仅涉及软件技术

知识,还要考虑潜在用户的心理感受因素和商品运营规律因素。

(3) 杀虫剂现象

1990 年,Boris Beizer 在其编著的《Software Testing Techniques》(第二版)中提到了“杀虫剂怪事”一词,同一种测试工具或方法用于测试同一类软件越多,则被测试软件对测试的免疫力就越强。这与农药杀虫是一样的,老用一种农药,则害虫就有了免疫力,农药就失去了作用。

由于软件开发人员在开发过程中可能碰见各种各样的主客观因素,再加上不可预见的突发性事件,所以再优秀的软件测试员采用一种测试方法或者工具也不可能检测出所有的缺陷。为了克服被测试软件的免疫力,软件测试员必须不断编写新的测试程序,对程序的各个部分进行不断地测试,以避免被测试软件对单一的测试程序具有免疫力而使软件缺陷不被发现。这就对软件测试人员的素质提出了很高的要求。

(4) 缺陷的不确定性

在软件测试中还有一个让人不容易判断的现象是缺陷的不确定性,即并不是所有的软件缺陷都需要被修复。对于究竟什么才算是软件缺陷是一个很难把握的标准,在任何一本软件测试的书中都只能给出一个笼统的定义。实际测试中需要把这一定义根据具体的被测对象明确化。即使这样,具体的测试人员对软件系统的理解不同,还是会出现不同的标准。

当确定是软件缺陷时,若出现以下情况,软件缺陷就不能被修复。

- 修复的风险太大。软件在编译期间,本身是一个很脆弱的系统,由于在整个软件系统中各个模块之间有着千丝万缕的联系,使得单一修复某一段代码可能会引起大量的未知的缺陷,所以在某些非常时期不修复反而是最保险的做法。
- 时间不够。在商业社会中,当部分软件缺陷没有足够的时间修复,就只能在说明书中列出可能出现的缺陷。
- 不会引起大的问题。为了防止整个系统由于局部修复而出现某些问题,在特殊情况下,不常出现的小问题可以暂时忽略。
- 可以理解成新的功能。某些特殊的缺陷有时从另一个方面看可以理解成一种新的功能,这是大多数商务软件在处理一些特殊缺陷时采取的做法。

2.1.2　软件测试的经济性

软件测试的经济性有两方面体现：一是体现在测试工作在整个项目开发过程中的重要地位,二是体现在应该按照什么样的原则进行测试,以实现测试成本与测试效果的统一。软件工程的总目标是充分利用有限的人力和物力资源,高效率、高质量地完成测试。结合 2.1.1 小节关于穷举测试具有不可行性,就可以理解为什么要在测试量与测试成本的曲线中选取最优测试点。为了降低测试成本,在选择测试用例时要遵守以下原则：

- 被测对象的测试等级应该取决于被测对象在整个软件开发项目中的重要地位和一旦发生故障会造成的损失情况来综合分析。
- 要制定科学有效的测试策略。在保证能够尽可能多地发现软件缺陷的前提下，尽量少地使用测试用例。如何找到最优测试点，掌握好测试用量是至关重要的。一位有经验的软件管理人员在谈到软件测试时曾这样说过："不充分的测试是愚蠢的，而过度的测试是一种罪孽"。测试不足意味着让用户承担隐藏错误带来的危险，过度测试则会浪费许多宝贵的资源。

测试是软件生命期中费用消耗最大的环节。测试费用除了测试的直接消耗外，还包括其他的相关费用。影响测试费用的主要因素有：

（1）软件面向的目标用户

软件产品需要达到的标准决定了测试的数量。对于那些至关重要的系统必须进行更多的测试。一台在 Boeing 757 上的系统应该比一个用于公共图书馆中检索资料的系统需要更多的测试。一个用来控制银行证券实时交易的系统应该比一个简单的网上实时交流系统具有更大的可靠性与可信度。一个用于国防的大型安全关键软件的开发组比一个网络游戏软件开发组要有苛刻得多的查找错误方面的要求。

（2）可能出现的用户数量

一个系统的目标用户数量的多少也在很大程度上影响了测试必要性的程度。这主要是由于用户团体在经济方面的影响。一个在全世界范围内有几千个用户的系统肯定比一个只在办公室中运行的有两三个用户的系统需要更多的测试。如果出现问题，前一个系统的经济影响肯定比后一个系统大。另外，在错误处理的分配上，所需花费代价的差别也很大。如果在内部系统中发现了一个严重的错误，处理错误的费用就会相对少一些。如果要处理一个遍布全世界的错误则要花费相当大的财力和精力，而且还会给开发公司造成严重的信誉危机和潜在用户的流失。

（3）潜在缺陷造成的影响

在考虑测试的必要性时，还需要将系统中所包含的信息价值考虑在内。例如一个支持许多家大银行或众多证券交易所的客户机/服务器系统中一定含有经济价值非常高的内容。因为由于银行证券系统的特殊性，一旦出现问题，影响的将不仅是银行或证券公司，错误将波及所有与银行或证券公司有业务往来的公司或个人，后果将非常恶劣。"很显然，这样的大型系统和其他单一的小型应用系统相比，需要进行更多的测试。这两种系统的用户都希望得到高质量、无错误的系统，但是前一种系统的影响比后一种要大得多。"因此我们应该从经济方面考虑，投入与经济价值相对应的时间和金钱去进行测试。

（4）开发机构的业务能力

一个没有标准和缺少经验的开发机构很可能会开发出充满错误的软件系统。而一个建立了标准和有很多经验的开发机构开发出来的软件系统中的错误将会少很多。然而，

那些需要进行大幅度改善的机构反而不大可能认识到自身的弱点。那些需要进行更加严格的软件测试的机构往往是最不可能进行这一活动的。在许多情况下,机构的管理部门并不能真正地理解开发一个高质量的软件系统的好处。反而是那些拥有很多经验和建立了严格标准的开发机构更加重视软件测试的重要性。

2.1.3 软件测试的充分性准则

软件测试的充分性准则有以下几点:

- 对任何软件都存在有限的充分测试集合。
- 当一个测试的数据集合对于一个被测的软件系统的测试是充分的,那么再多增加一些测试数据仍然是充分的。这一特性称为软件测试的单调性。
- 即使对软件所有成分都进行了充分的测试,也并不意味着整个软件的测试已经充分了。这一特性称为软件测试的非复合性。
- 即使对一个软件系统整体的测试是充分的,也并不意味着软件系统中各个成分都已经充分地得到了测试。这个特性称为软件测试的非分解性。
- 软件测试的充分性与软件的需求、软件的实现都相关。
- 软件测试的数据量正比于软件的复杂度。这一特性称为软件测试的复杂性。
- 随着测试次数的增加,检查出软件缺陷的几率随之不断减少。软件测试具有回报递减率。

2.1.4 软件测试的误区

随着软件产业工业化、模块化地发展,在软件开发组中软件测试人员的重要性也不断地突出。在国外,很多著名企业早已对软件测试工作十分重视。比如著名的微软公司,其软件测试人员与开发人员的比例已经达到 2∶1。可见软件测试对于一个软件开发项目的成功与否具有十分重要的意义。但是在实际的项目开发与管理中仍然存在很多管理上或者技术上的误区。

(1) 期望用测试自动化代替大部分人工劳动

通过应用自动化测试工具能够帮助完成部分重复枯燥的手工作业,但自动化测试工具不能完全代替人工测试。一般来讲,产品化的软件较适于功能测试的自动化,而由标准模块组装的系统更适合功能测试的自动化,这是因为这类软件功能稳定,界面变化不大。

对于测试自动化的使用可以按以下规则来进行:自动化 20%的测试用例,用于覆盖 80%的用户操作密集的功能和核心商业逻辑(例如工资计算准确度要求高,虽然每月才执行一次)。实现功能测试自动化来完成重复枯燥的回归测试任务,引入性能测试自动化工具来改善测试的广度和深度。自动化会带来一点好处,毕竟机器和脚本是客观的,它总是会完成测试员所分配的所有任务,而没有半点遗漏,从而有助于测试员真正掌握和控制回归测试的覆盖率。

(2) 忽视需求阶段的参与

在某些公司，产品原始需求文档本来就不是很完善，从市场调研人员到项目经理、开发经理，开发小组组长，再到具体编写代码的程序员，每一层之间的传递都有可能存在需求理解上的偏差。让测试人员参与需求阶段的工作，可以在一定程度上起到双保险，更好地杜绝需求和实现之间差异的发生。软件测试工作同时兼顾了“证明软件的实现和需求是一致的”和“验证软件在某些情况下可能会产生问题”两个方面。因此，测试人员对需求的理解就从另一个角度影响了整个测试工作的可靠性和效率。测试人员和开发人员同时、同等地从上游获得需求，并持有自己的理解，可以排除部分功能实现和需求错位的问题。

(3) 软件测试是技术要求不高的岗位

单从目前用人最多的黑盒测试岗位来说，要求测试人员对计算机技术的精通能力或许并不是很高。实际上，测试人员除了逻辑思维、沟通能力等自身素质外，技能暂且可以分为两种：一种是行业知识，比如丰富的财务或 ERP 实施经验；另一种是计算机技术，比如计算机语言程序设计和软件项目开发经验。

好的测试人员，不但要不懈地执行常规的测试任务，更要有严谨的态度和缜密的思维，去覆盖更多的“可能”，发现别人很难找到的软件缺陷。要利用自己丰富的行业经验，判断从需求到系统功能的实现是否合理。要站在一定高度对软件框架、设计方法、项目管理等做出合理的建议。

所有这些，加上软件测试管理相关的其他技术(如配置管理等)，对于一名合格的软件测试人员的素质要求是很高的。

2.2 软件测试的流程

2.2.1 软件开发的 V 模型

1. V 模型

软件测试是有阶段性的，而软件测试的流程与软件设计周期究竟是什么样的关系呢？关于软件开发流程的 V 模型是一个广为人知的模型，如图 2-2 所示。在 V 模型中，从左到右描述了基本的开发过程和测试行为，为软件的开发人员和测试管理者提供了一个极为简单的框架。V 模型的价值在于它非常明确地标明了测试过程中存在的不同级别，并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系。

在 V 模型中各个测试阶段的执行流程是：单元测试是基于代码的测试，最初由开发人员执行，以验证其可执行程序代码的各个部分是否已达到了预期的功能要求；集成测试验证了两个或多个单元之间的集成是否正确，并且有针对性地对详细设计中所定义的各单元之间的接口进行检查；在单元测试和集成测试完成之后，系统测试开始用客户环

境模拟系统的运行，以验证系统是否达到了在概要设计中所定义的功能和性能；最后，当技术部门完成了所有测试工作，由业务专家或用户进行验收测试，以确保产品能真正符合用户业务上的需要。图 2-2 描绘出了各个测试环节在整个软件测试工作中的相互联系与制约关系。

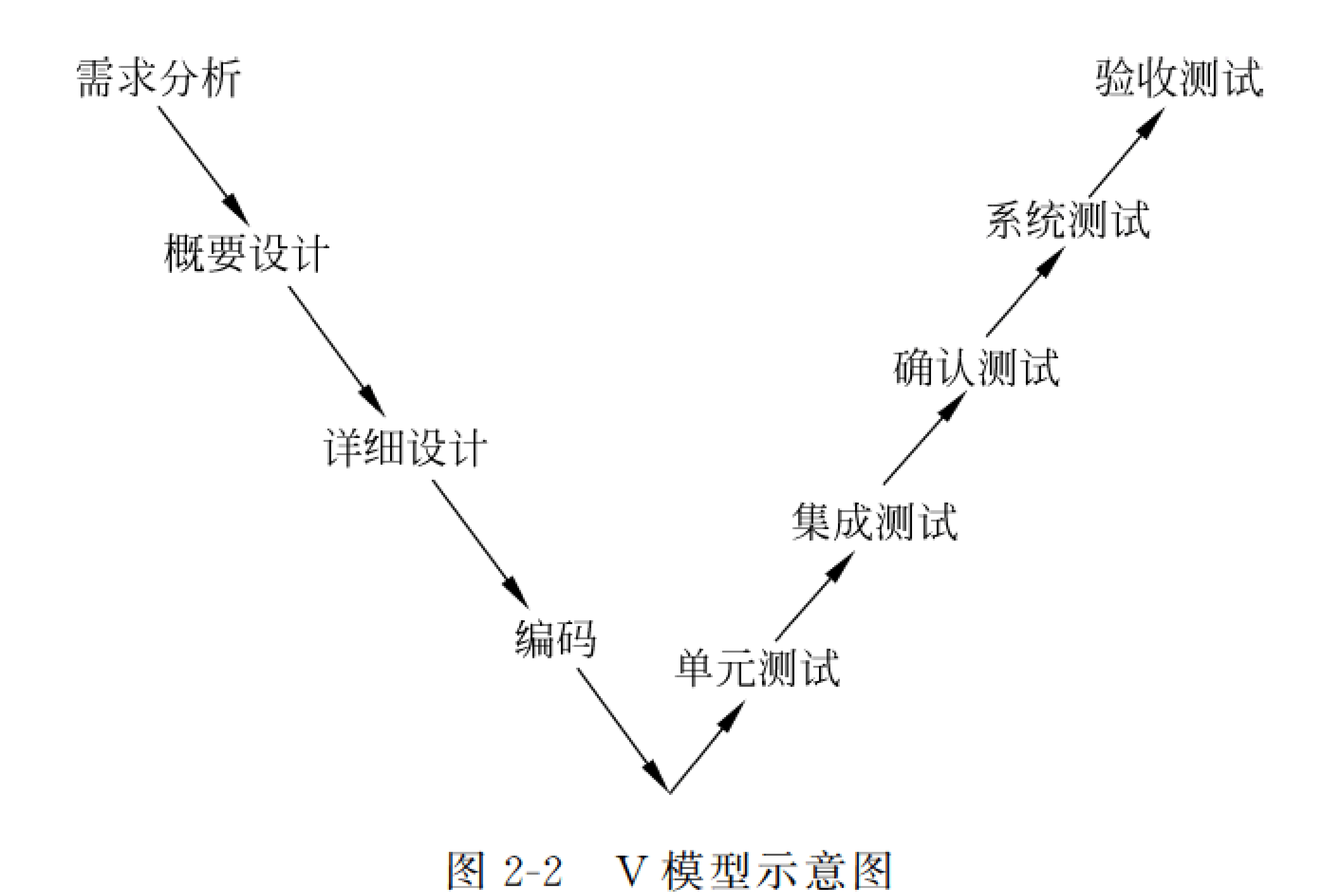


图 2-2　V 模型示意图

2. 软件测试过程

软件测试过程按各测试阶段的先后顺序可分为单元测试、集成测试、确认（有效性）测试、系统测试和验收（用户）测试 5 个阶段，如图 2-3 所示。

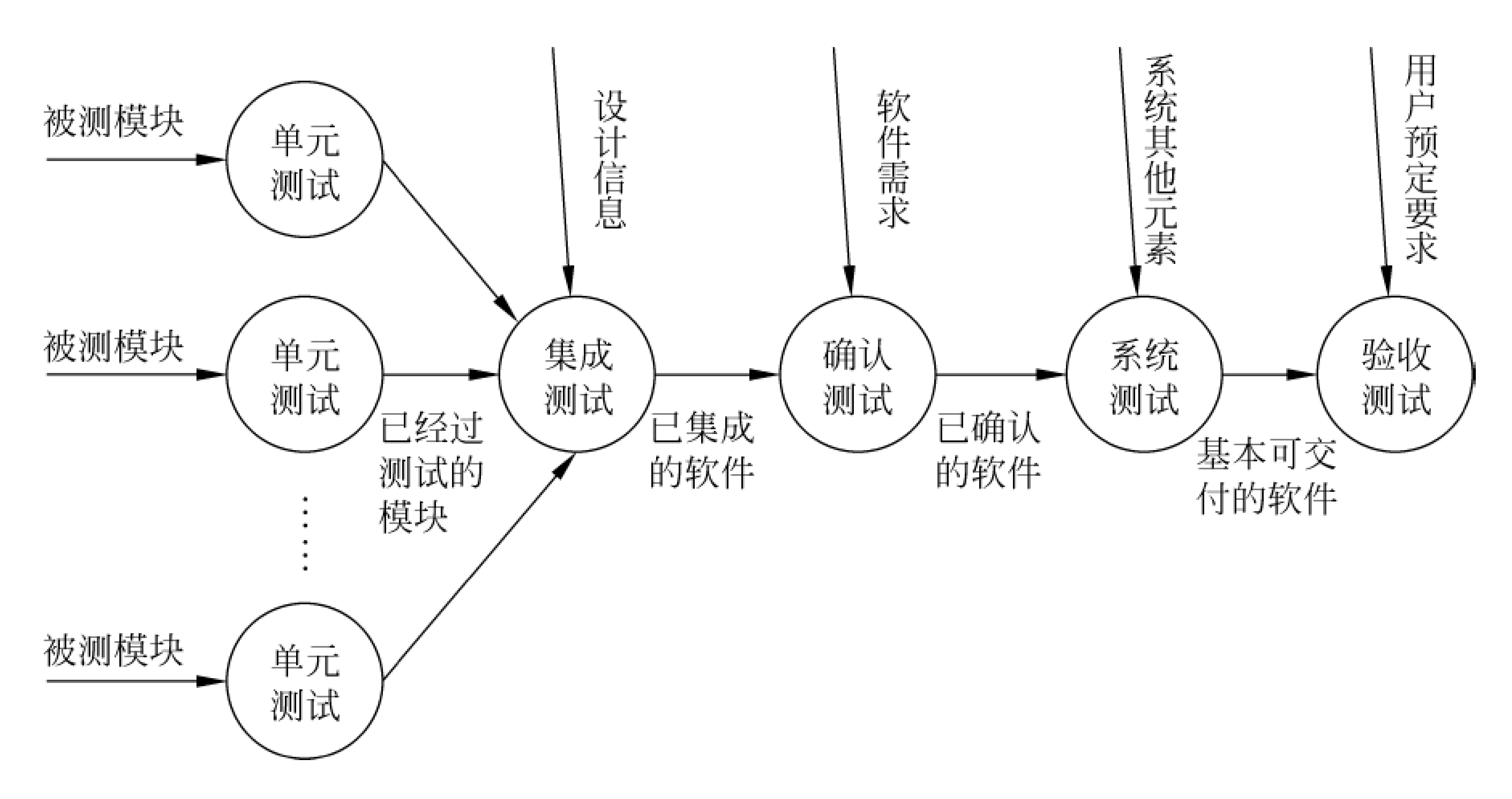


图 2-3　测试各阶段示意图

（1）单元测试：测试执行的开始阶段。测试对象是每个单元，测试目的是保证每个模块或组件能正常工作。单元测试主要采用白盒测试方法，检测程序的内部结构。

(2) 集成测试：也称组装测试。在单元测试基础上，对已测试过的模块进行组装，进行集成测试。测试目的是检验与接口有关的模块之间的问题。集成测试主要采用黑盒测试方法。

(3) 确认测试：也称有效性测试。在完成集成测试后，验证软件的功能和性能及其他特性是否符合用户要求。测试目的是保证系统能够按照用户预定的要求工作。确认测试通常采用黑盒测试方法。

(4) 系统测试：在完成确认测试后，为了检验它能否与实际环境（如软硬件平台、数据和人员等）协调工作，还需要进行系统测试。可以说，系统测试之后，软件产品基本满足开发要求。

(5) 验收测试：测试过程的最后一个阶段。验收测试主要突出用户的作用，同时软件开发人员也应该参与进去。

软件测试阶段的输入信息包括两类：

- 软件配置：指测试对象。通常包括需求说明书、设计说明书和被测试的源程序等；
- 测试配置：通常包括测试计划、测试步骤、测试用例以及具体实施测试的测试程序、测试工具等。

对测试结果与预期的结果进行比较以后，即可判断是否存在错误，决定是否进入排错阶段，进行调试任务。对修改以后的程序要进行重新测试，因为修改可能会带来新的问题。

通常根据出错的情况得到出错率来预估被测软件的可靠性，这将对软件运行后的维护工作有重要价值。

2.2.2 单元测试

1. 单元测试的定义

单元测试（Unit Testing）是对软件基本组成单元进行的测试。单元测试的对象是软件设计的最小单位——模块。很多人将单元的概念误解为一个具体函数或一个类的方法，这种理解并不准确。作为一个最小的单元应该有明确的功能定义、性能定义和接口定义，而且可以清晰地与其他单元区分开来。一个菜单、一个显示界面或者能够独立完成的具体功能都可以是一个单元。从某种意义上单元的概念已经扩展为组件（component）。

单元测试通常是开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。通常而言，一个单元测试是用于判断某个特定条件（或者场景）下某个特定函数的行为。例如，可以把一个很大的值放入一个有序表中去，然后确认该值出现在有序表的尾部。或者从字符串中删除匹配某种模式的字符，然后确认字符串确实不再包含这些字符了。单元测试是由程序员自己来完成，最终受益的也是程序员自己。可以这

么说，程序员有责任编写功能代码，同时也就有责任为自己的代码编写单元测试。执行单元测试，就是为了证明这段代码的行为和期望的一致。打一个比喻，单元测试就像工厂在组装一台电视机之前，会对每个元件都进行测试，这就是单元测试。其实我们每天都在做单元测试。你写了一个函数，除了极简单的外，总是要执行一下，看看软件的功能是否正常，有时还要想办法输出些数据，比如弹出信息窗口等，这也是单元测试。一般把这种单元测试称为临时单元测试。对于程序员来说，如果养成了对自己写的代码进行单元测试的习惯，不但可以写出高质量的代码，而且还能提高编程水平。

2. 单元测试的目标

单元测试的主要目标是确保各单元模块被正确地编码。单元测试除了保证测试代码的功能性，还需要保证代码在结构上具有可靠性和健全性，并且能够在所有条件下正确响应。进行全面的单元测试，可以减少应用级别所需的工作量，并且彻底减少系统产生错误的可能性。如果手动执行，单元测试可能需要大量的工作，自动化测试会提高测试效率。

3. 单元测试的内容

单元测试的主要内容有：模块接口测试；局部数据结构测试；独立路径测试；出错处理测试；边界条件测试。如图 2-4 所示，这些测试都作用于模块，共同完成单元测试任务。

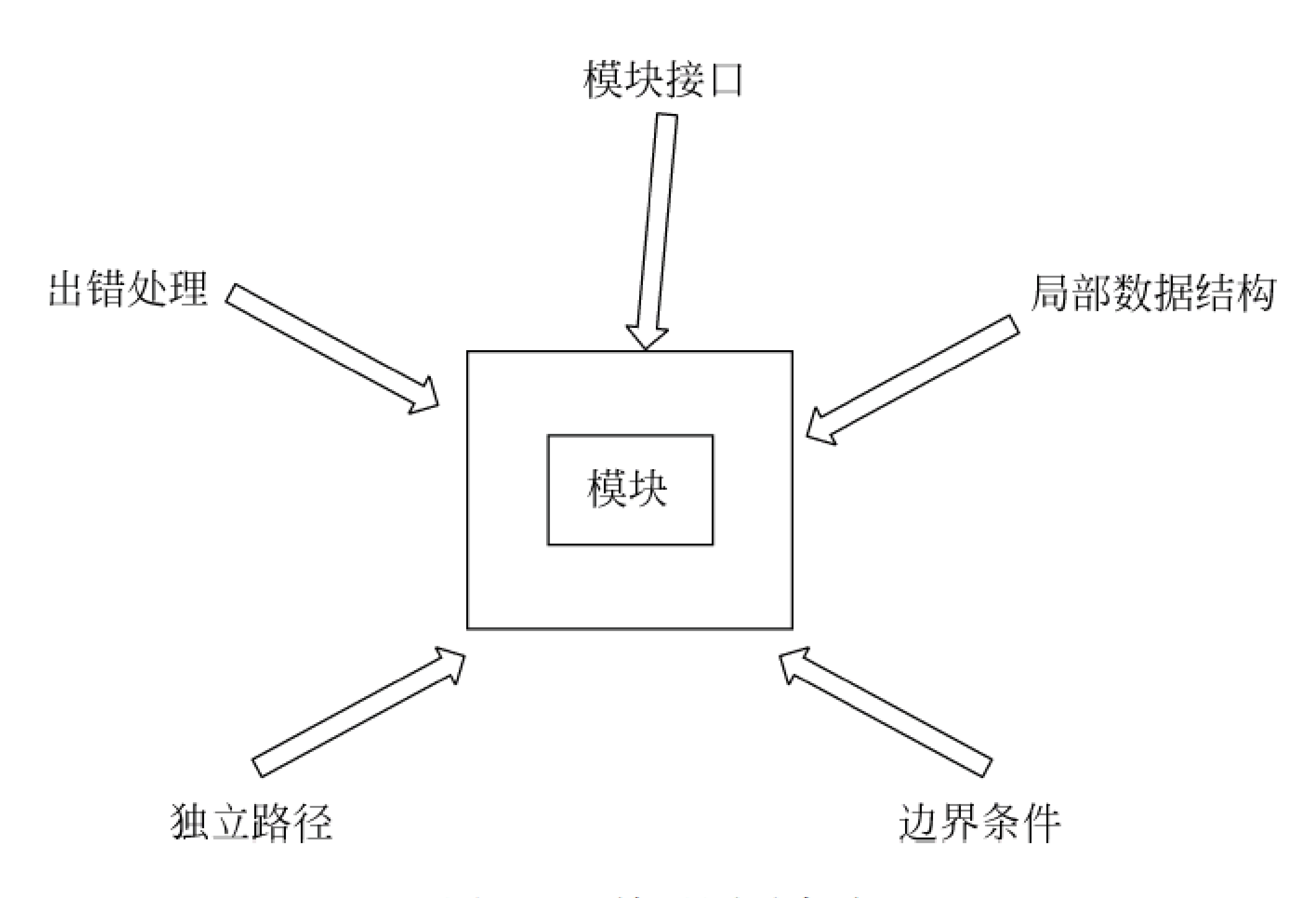


图 2-4　单元测试任务

- 模块接口测试：对通过被测模块的数据流进行测试。为此，对模块接口，包括参数表、调用子模块的参数、全程数据、文件输入/输出操作都必须检查。
- 局部数据结构测试：设计测试用例检查数据类型说明、初始化、默认值等方面的问题，还要查清全程数据对模块的影响。
- 独立路径测试：选择适当的测试用例，对模块中重要的执行路径进行测试。基本

路径测试和循环测试可以发现大量的路径错误，是最常用且最有效的测试技术。

- 出错处理测试：检查模块的错误处理功能是否包含有错误或缺陷。例如，是否拒绝不合理的输入；出错的描述是否难以理解、是否对错误定位有误、是否出错原因报告有误、是否对错误条件的处理不正确；在对错误处理之前错误条件是否已经引起系统的干预等。
- 边界条件测试：要特别注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，认真加以测试。此外，如果对模块运行时间有要求，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响模块运行时间的因素。这类信息对进行性能评价是十分有用的。

4. 单元测试的步骤

通常单元测试在编码阶段进行。当源程序代码编制完成，经过评审和验证，确认没有语法错误之后，就开始进行单元测试的测试用例设计。利用设计文档，设计可以验证程序功能、找出程序错误的多个测试用例。对于每一组输入，应有预期的正确结果。

模块接口测试中的被测模块并不是一个独立的程序，在考虑测试模块时，同时要考虑它和外界的联系，用一些辅助模块去模拟与被测模块相关联的其他模块。这些辅助模块可分为两种：

(1) 驱动模块(driver)：相当于被测模块的主程序。它接收测试数据，把这些数据传送给被测模块，最后输出实测结果。

(2) 桩模块(stub)：用以代替被测模块调用的子模块。桩模块可以做少量的数据操作，不需要把子模块所有功能都带进来，但不允许什么事情也不做。

被测模块、与它相关的驱动模块以及桩模块共同构成了一个“测试环境”，如图 2-5 所示。

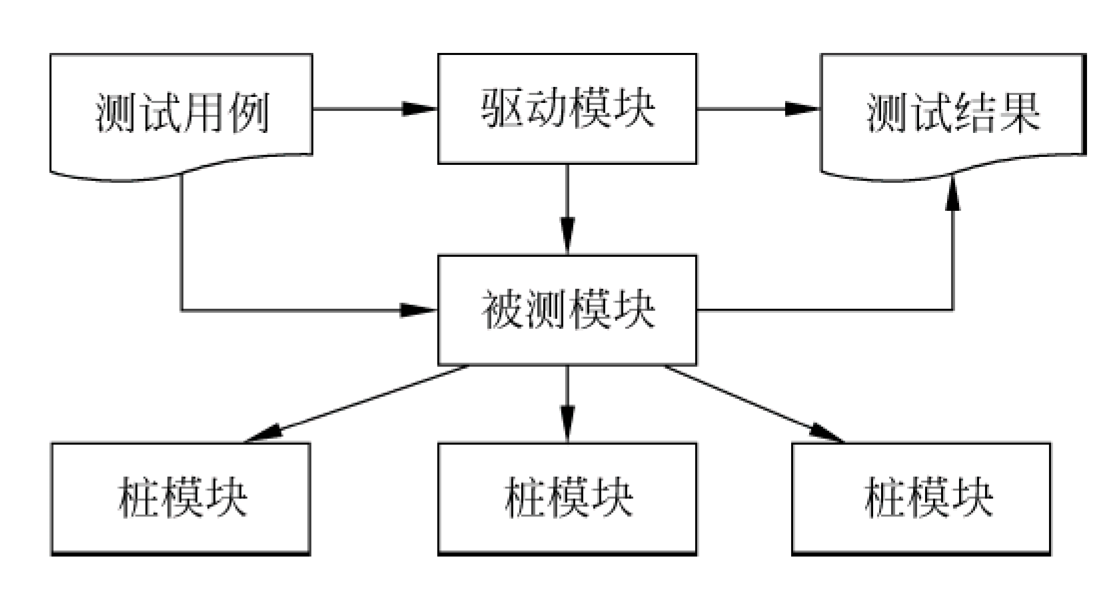


图 2-5　单元测试环境

如果一个模块要完成多种功能，并且以程序包或对象类的形式出现，例如 Ada 中的包，MODULA 中的模块，C++ 中的类，这时可以将模块看成由几个小程序组成。对其中

的每个小程序先进行单元测试要做的工作，对关键模块还要做性能测试。对支持某些标准规程的程序，更要着手进行互联测试。有人把这种情况特别称为模块测试，以区别单元测试。

5. 采用单元测试的原因

程序员编写代码时，一定会反复调试保证其能够编译通过。如果是编译没有通过的代码，没有任何人会愿意交付给自己的老板。但代码通过编译，只是说明了它的语法正确，程序员却无法保证它的语义也一定正确。没有任何人可以轻易承诺这段代码的行为一定是正确的，单元测试这时会为此做出保证。编写单元测试就是用来验证这段代码的行为是否与软件开发人员期望的一致。有了单元测试，程序员可以自信地交付自己的代码，而没有任何的后顾之忧。

什么时候进行单元测试呢？单元测试越早越好。早到什么程度呢？开发理论讲究TDD（测试驱动开发），即先编写测试代码，再进行开发。在实际的工作中，可以不必过分强调先干什么后干什么，重要的是高效和感觉舒适。从实际开发经验来看，先编写产品函数的框架，然后编写测试函数，针对产品函数的功能编写测试用例，然后编写产品函数的代码，每写一个功能点都运行测试，随时补充测试用例。所谓先编写产品函数的框架，是指先编写函数空的实现，有返回值的随便返回一个值，编译通过后再编写测试代码。这时，函数名、参数表、返回类型都应该确定下来了，所编写的测试代码以后需修改的可能性比较小。

由谁来完成单元测试呢？单元测试与其他测试不同，单元测试可看作是编码工作的一部分，应该由程序员完成。也就是说，经过了单元测试的代码才是已完成的代码，提交产品代码时也要同时提交测试代码。测试部门可以进行一定程度的审核。在传统的结构化编程语言中，比如 C 语言，要进行测试的单元一般是函数或子过程。在像 C++ 这样的面向对象的语言中，要进行测试的基本单元是类。对 Ada 语言来说，开发人员可以选择是在独立的过程和函数，还是在 Ada 包的级别上进行单元测试。单元测试的原则同样被扩展到第四代语言（4GL）的开发中，在这里基本单元被典型地划分为一个菜单或显示界面。单元测试是作为无错编码的一种辅助手段，在一次性的开发过程中使用。另外，单元测试必须是可重复的，无论是在软件修改，还是移植到新的运行环境的过程中。因此，所有的测试都必须在整个软件系统的生命周期中进行维护。

通过单元测试，测试人员可以验证开发人员所编写的代码是按照先前设想的方式进行的，输出结果符合预期值，这就实现了单元测试的目的。与后面的测试相比，单元测试创建简单，维护容易，并且可以更方便地重复。《实用软件度量》（Capers Jones，McGraw-Hill，1991）列出了准备测试、执行测试和修改缺陷所花费的时间（以一个功能点为基准），这些测试显示出了单元测试的成本效率大约是集成测试的两倍、系统测试的三倍，如图 2-6 所示。术语域测试是指软件在投入使用后，针对某个领域所作的所有测试活动。

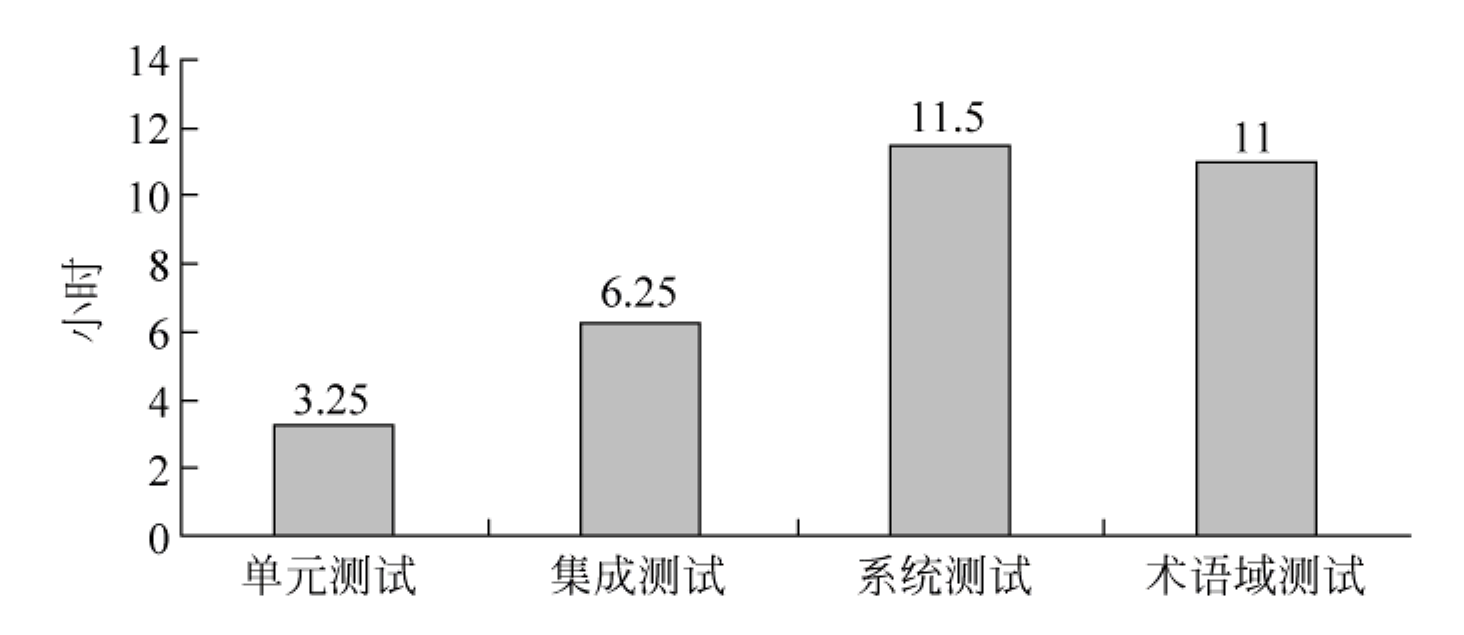


图 2-6　各测试阶段发现缺陷的耗时

2.2.3　集成测试

1. 集成测试的定义

在完成单元测试的基础上，需要将所有模块按照设计要求组装成为系统。这时需要考虑以下问题：

- 在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；
- 一个模块的功能是否会对另一个模块的功能产生不利的影响；
- 各个子功能组合起来，能否达到预期要求的父功能；
- 全局数据结构是否有问题；
- 单个模块的误差累积起来，是否会放大，从而达到不能接受的程度；
- 单个模块的错误是否会导致数据库错误。

集成测试(Integration Testing)是介于单元测试和系统测试之间的过渡阶段，与软件开发计划中的软件概要设计阶段相对应，是单元测试的扩展和延伸。集成测试的定义是根据实际情况对程序模块采用适当的集成测试策略组装起来，对系统的接口以及集成后的功能进行正确校验的测试工作。集成测试也称为综合测试。实践表明，软件的一些模块能够单独地工作，但并不能保证连接之后也肯定能正常工作。程序在某些局部反映不出来的问题，在全局上有可能暴露出来，影响软件功能的实现。所以，集成测试是针对程序整体结构的测试。

2. 集成测试的层次

软件的开发过程是一个从需求分析到概要设计、详细设计以及编码实现的逐步细化的过程，那么单元测试到集成测试再到系统测试就是一个逆向求证的过程。集成测试内部对于传统软件和面向对象的应用系统有两种层次的划分。

对于传统软件来讲，可以把集成测试划分为三个层次：

- 模块内集成测试；
- 子系统内集成测试；

• 子系统间集成测试。

对于面向对象的应用系统来说，可以把集成测试分为两个阶段：

• 类内集成测试；
• 类间集成测试。

3. 集成测试的模式

选择什么方式把模块组装起来形成一个可运行的系统，直接影响到模块测试用例的形式、所用测试工具的类型、模块编号的次序和测试的次序、生成测试用例的费用和调试的费用。集成测试模式是软件集成测试中的策略体现，其重要性是明显的，直接关系到软件测试的效率、结果等，一般是根据软件的具体情况来决定采用哪种模式。通常，把模块组装成为系统的测试方式有两种：

(1) 一次性集成测试方式(No-Incremental Integration)

一次性集成测试方式也称作非增值式集成测试。先分别测试每个模块，再把所有模块按设计要求放在一起结合成所需要实现的程序。

如图 2-7 所示是按照一次性集成测试方式的实例。图 2-7(a)表示的是整个系统结构，共包含 6 个模块。具体测试过程如下：

• 为模块 B 配备驱动模块 D1，来模拟模块 A 对 B 的调用；为模块 B 配备桩模块 S1，来模拟模块 E 被 B 调用。对模块 B 进行单元测试，如图 2-7(b)所示。
• 为模块 D 配备驱动模块 D3，来模拟模块 A 对 D 的调用。为模块 D 配备桩模块 S2，来模拟模块 F 被 D 调用。对模块 D 进行单元测试，如图 2-7(d)所示。
• 为模块 C、E、F 分别配备驱动模块 D2、D4、D5，对模块 C、E、F 分别进行单元测试，如图 2-7(c)、图 2-7(e)、图 2-7(f)所示。

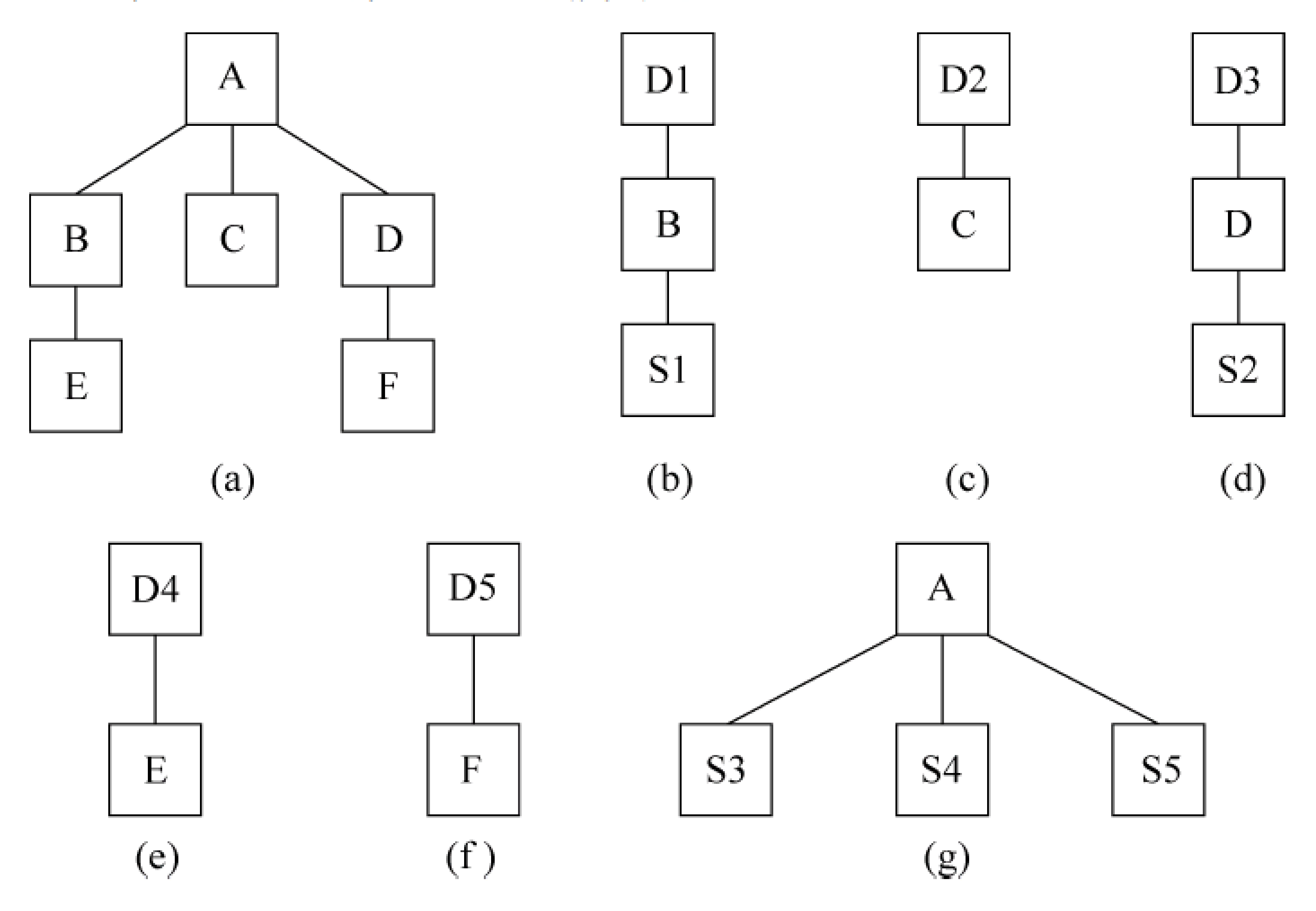


图 2-7　一次性集成测试方式

- 为主模块 A 配备三个桩模块 S3、S4、S5，对模块 A 进行单元测试，如图 2-7(g)所示。
- 在将模块 A、B、C、D、E 分别进行了单元测试之后，再一次性进行集成测试。
- 测试结束。

(2) 增值式集成测试方式

把下一个要测试的模块同已经测好的模块结合起来进行测试，测试完毕，再把下一个应该测试的模块结合进来继续进行测试。在组装的过程中边连接边测试，以发现连接过程中产生的问题。通过增值逐步组装成为预先要求的软件系统。增值式集成测试方式有以下三种。

① 自顶向下增值测试方式(Top-down Integration)

主控模块作为测试驱动，所有与主控模块直接相连的模块作为桩模块；根据集成的方式(深度或广度)，每次用一个模块把从属的桩模块替换成真正的模块；在每个模块被集成时，都必须已经进行了单元测试；进行回归测试以确定集成新模块后没有引入错误。这种组装方式将模块按系统程序结构，沿着控制层次自顶向下进行组装。自顶向下的增值方式在测试过程中较早地验证了主要的控制和判断点。选用按深度方向组装的方式，可以首先实现和验证一个完整的软件功能。

图 2-8 表示的是按照深度优先方式遍历的自顶向下增值的集成测试实例。具体测试过程如下：

- 在树状结构图中，按照先左后右的顺序确定模块集成路线。
- 如图 2-8(a)所示，先对顶层的主模块 A 进行单元测试，就是对模块 A 配以桩模块 S1、S2 和 S3，用来模拟它所实际调用的模块 B、C、D，然后进行测试。
- 如图 2-8(b)所示，用实际模块 B 替换掉桩模块 S1，与模块 A 连接，再对模块 B 配以桩模块 S4，用来模拟模块 B 对 E 的调用，然后进行测试。
- 图 2-8(c)是将模块 E 替换掉桩模块 S4 并与模块 B 相连，然后进行测试。
- 判断模块 E 没有叶子结点，也就是说以 A 为根结点的树状结构图中的最左侧分支深度遍历结束，转向下一个分支。
- 如图 2-8(d)所示，模块 C 替换掉桩模块 S2，连到模块 A 上，然后进行测试。
- 判断模块 C 没有桩模块，转到树状结构图的最后一个分支。
- 如图 2-8(e)所示，模块 D 替换掉桩模块 S3，连到模块 A 上，同时给模块 D 配以桩模块 S5，来模拟其对模块 F 的调用，然后进行测试。
- 如图 2-8(f)所示，去掉桩模块 S5，替换成实际模块 F 连接到模块 D 上，然后进行测试。
- 对树状结构图进行了完全测试，测试结束。

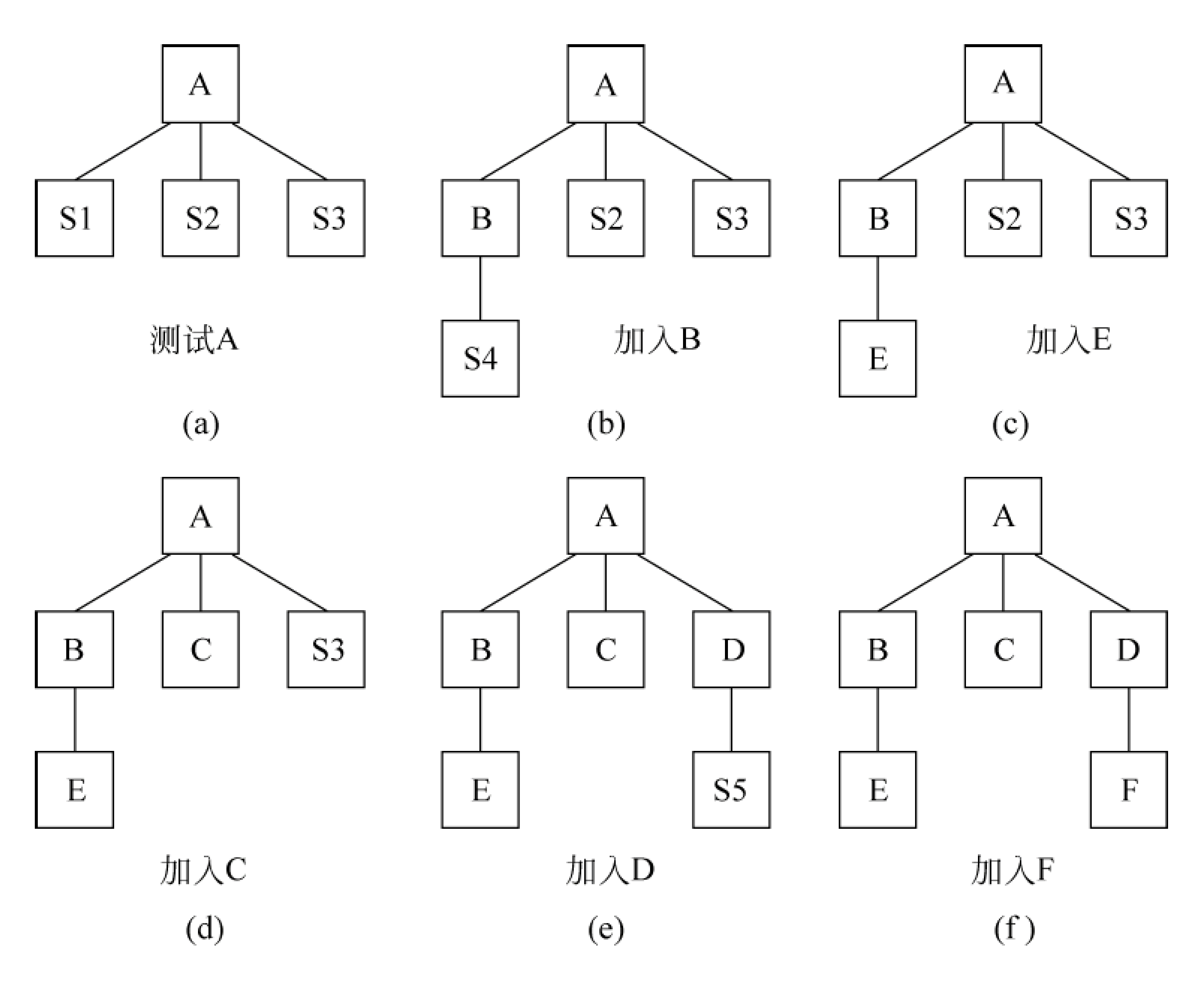


图 2-8　自顶向下增值测试方式

② 自底向上增值测试方式(Bottom-up Integration)

组装从最底层的模块开始,组合成一个构件,用以完成指定的软件子功能。编制驱动程序,协调测试用例的输入与输出;测试集成后的构件;按程序结构向上组装测试后的构件,同时除掉驱动程序。这种组装的方式是从程序模块结构的最底层模块开始组装和测试。因为模块是自底向上进行组装,对于一个给定层次的模块,它的子模块(包括子模块的所有下属模块)已经组装并测试完成,所以不再需要桩模块。在模块的测试过程中如果需要从子模块得到信息时可以直接运行子模块获得。

图 2-9 表示的是按照自底向上增值的集成测试例子。首先,对处于树状结构图中叶子结点位置的模块 E、C、F 进行单元测试,如图 2-9(a)、图 2-9(b)和图 2-9(c)所示,分别配以驱动模块 D1、D2 和 D3,用来模拟模块 B、模块 A 和模块 D 对它们的调用。然后,如图 2-9(d)和图 2-9(e)所示,去掉驱动模块 D1 和 D3,替换成模块 B 和 D 分别与模块 E 和 F 相连,并且设立驱动模块 D4 和 D5 进行局部集成测试。最后,如图 2-9(f)所示,对整个系统结构进行集成测试。

③ 混合增值测试方式(Modified Top-down Integration)

自顶向下增值的方式和自底向上增值的方式各有优缺点。

自顶向下增值方式的缺点是需要建立桩模块。要使桩模块能够模拟实际子模块的功能是十分困难的,同时涉及复杂算法。真正输入/输出的模块处在底层,它们是最容易出

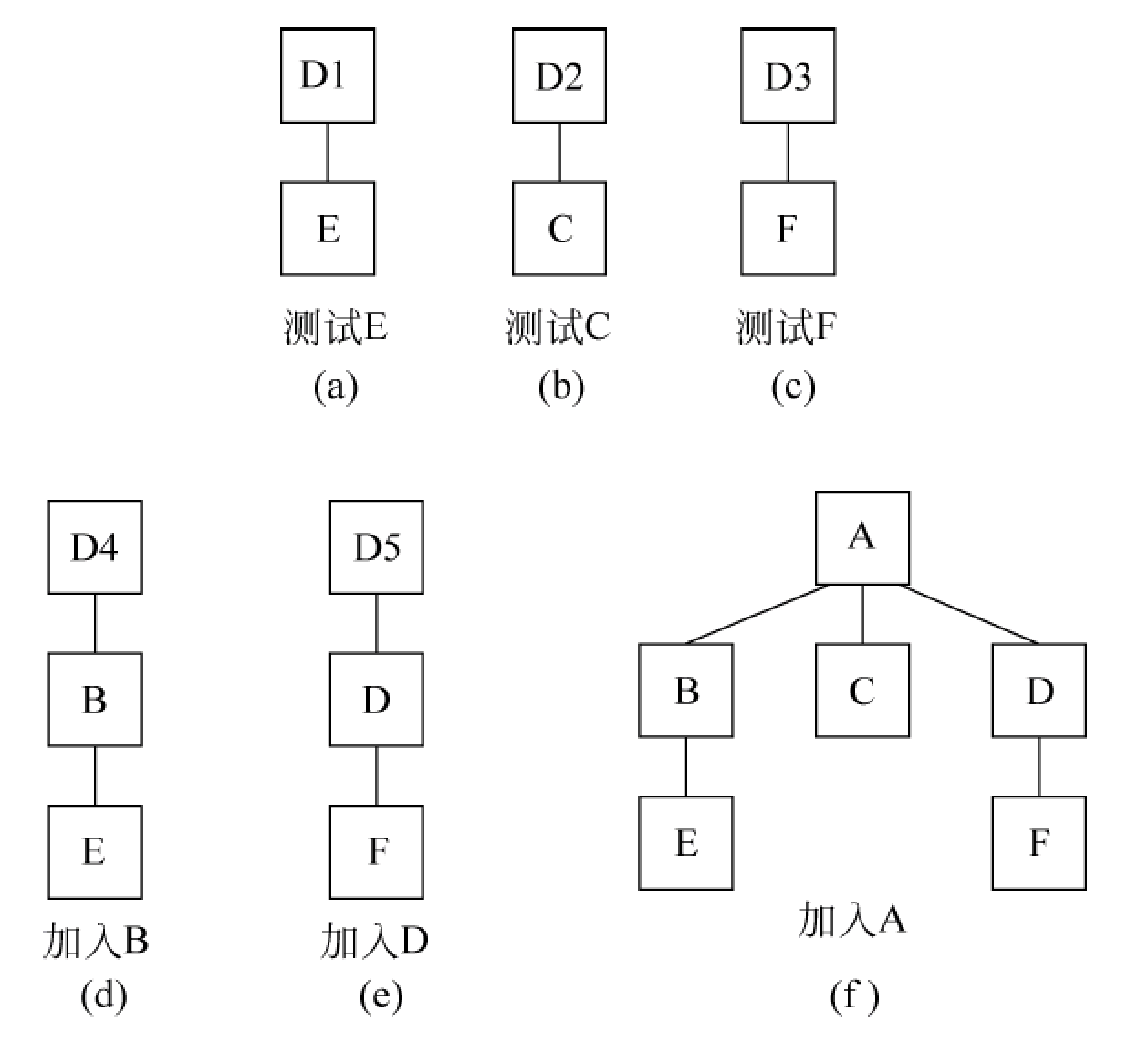


图 2-9 自底向上增值测试方式

问题的模块,并且直到组装和测试的后期才遇到这些模块,一旦发现问题,会导致过多的回归测试。

自顶向下增值方式的优点是能够较早地发现在主要控制方面存在的问题。

自底向上增值方式的缺点是"程序一直未能作为一个实体存在,直到最后一个模块加上去后才形成一个实体"。就是说,在自底向上组装和测试的过程中,对主要的控制直到最后才接触到。

自底向上增值方式的优点是不需要桩模块,建立驱动模块一般比建立桩模块容易,同时由于涉及复杂算法和真正输入/输出的模块最先得到组装和测试,可以把最容易出问题的部分在早期解决。此外自底向上增值的方式可以实施多个模块的并行测试。

有鉴于此,通常是把以上两种方式结合起来进行组装和测试。

- 改进的自顶向下增值测试:基本思想是强化对输入/输出模块和引入新算法模块的测试,并自底向上组装成为功能相当完整且相对独立的子系统,然后由主模块开始自顶向下进行增值测试。
- 自底向上—自顶向下的增值测试(混和法):首先对含读操作的子系统自底向上直至根结点模块进行组装和测试,然后对含写操作的子系统做自顶向下的组装与测试。
- 回归测试:这种方式采取自顶向下的方式测试被修改的模块及其子模块,然后将这一部分视为子系统,再自底向上测试,以检查该子系统与其上级模块的接口是否适配。

(3) 一次性集成测试方式与增值式集成测试方式的比较

- 增值式集成方式需要编写的软件较多,工作量较大,花费的时间较多,一次性集成方式的工作量较小;
- 增值式集成方式发现问题的时间比一次性集成方式早;
- 增值式集成方式比一次性集成方式更容易判断出问题的所在,因为出现的问题往往和最后加进来的模块有关;
- 增值式集成方式测试得更为彻底;
- 使用一次性集成方式可以多个模块并行测试。

这两种模式各有利弊,在时间条件允许的情况下采用增值式集成测试方式有一定的优势。

(4) 集成测试的组织和实施

集成测试是一种正规测试过程,必须精心计划,并与单元测试的完成时间协调起来。在制定测试计划时,应考虑如下因素:

- 采用何种系统组装方法来进行组装测试;
- 组装测试过程中连接各个模块的顺序;
- 模块代码编制和测试进度是否与组装测试的顺序一致;
- 测试过程中是否需要专门的硬件设备。

解决了上述问题之后,就可以列出各个模块的编制、测试计划表,标明每个模块单元测试完成的日期、首次集成测试的日期、集成测试全部完成的日期,以及需要的测试用例和所期望的测试结果。

在缺少软件测试所需要的硬件设备时,应检查该硬件的交付日期是否与集成测试计划一致。例如,若测试需要数字化仪和绘图仪,则相应测试应安排在这些设备能够投入使用之时,并需要为硬件的安装和交付使用保留一段时间,以留下时间余量。此外,在测试计划中需要考虑测试所需软件(驱动模块、桩模块、测试用例生成程序等)的准备情况。

4. 集成测试完成的标志

判定集成测试过程是否完成,可按以下几个方面检查:

- 成功地执行了测试计划中规定的所有集成测试;
- 修正了所发现的错误;
- 测试结果通过了专门小组的评审。

集成测试应由专门的测试小组来进行,测试小组由有经验的系统设计人员和程序员组成。整个测试活动要在评审人员出席的情况下进行。在完成预定的组装测试工作之后,测试小组应负责对测试结果进行整理、分析,形成测试报告。测试报告中要记录实际的测试结果、在测试中发现的问题、解决这些问题的方法以及解决之后再次测试的结果。此外还应提出目前不能解决、还需要管理人员和开发人员注意的一些问题,提供测试评审

和最终决策，以提出处理意见。集成测试需要提交的文档有：集成测试计划、集成测试规格说明、集成测试分析报告。

5. 采用集成测试的原因

所有的软件项目都不能摆脱系统集成这个阶段。不管采用什么开发模式，具体的开发工作总得从一个一个的软件单元做起，软件单元只有经过集成才能形成一个有机的整体。具体的集成过程可能是显性的也可能是隐性的。只要有集成，总是会出现一些常见问题，工程实践中，几乎不存在软件单元组装过程中不出任何问题的情况。集成测试需要花费的时间远远超过单元测试，直接从单元测试过渡到系统测试是非常危险的做法，可能使整个软件开发项目所耗费的时间成倍的增加。集成测试的必要性还在于一些模块虽然能够单独地工作，但并不能保证连接起来也能正常工作。程序在某些局部反映不出来的问题，有可能在全局上会暴露出来，影响功能的实现。

2.2.4 确认测试

1. 确认测试的定义

集成测试完成以后，分散开发的模块被连接起来，构成完整的程序，其中各模块之间接口存在的种种问题都已消除。于是测试工作进入确认测试(Validation Testing)阶段。

什么是确认测试，说法众多，其中最简明、最严格的解释是检验所开发的软件是否能按用户提出的要求运行，若能达到这一要求，则认为开发的软件是合格的。因而有的软件开发部门把确认测试称为合格性测试(Qualification Testing)。这里所说的客户要求通常指的是在软件规格说明书中确定的软件功能和技术指标，或是专门为测试所规定的确认准则。在确认测试阶段需要做的工作如图 2-10 所示。首先要进行有效性测试以及软件配置审查，然后进行验收测试和安装测试，在通过了专家鉴定之后，才能成为可交付的软件。

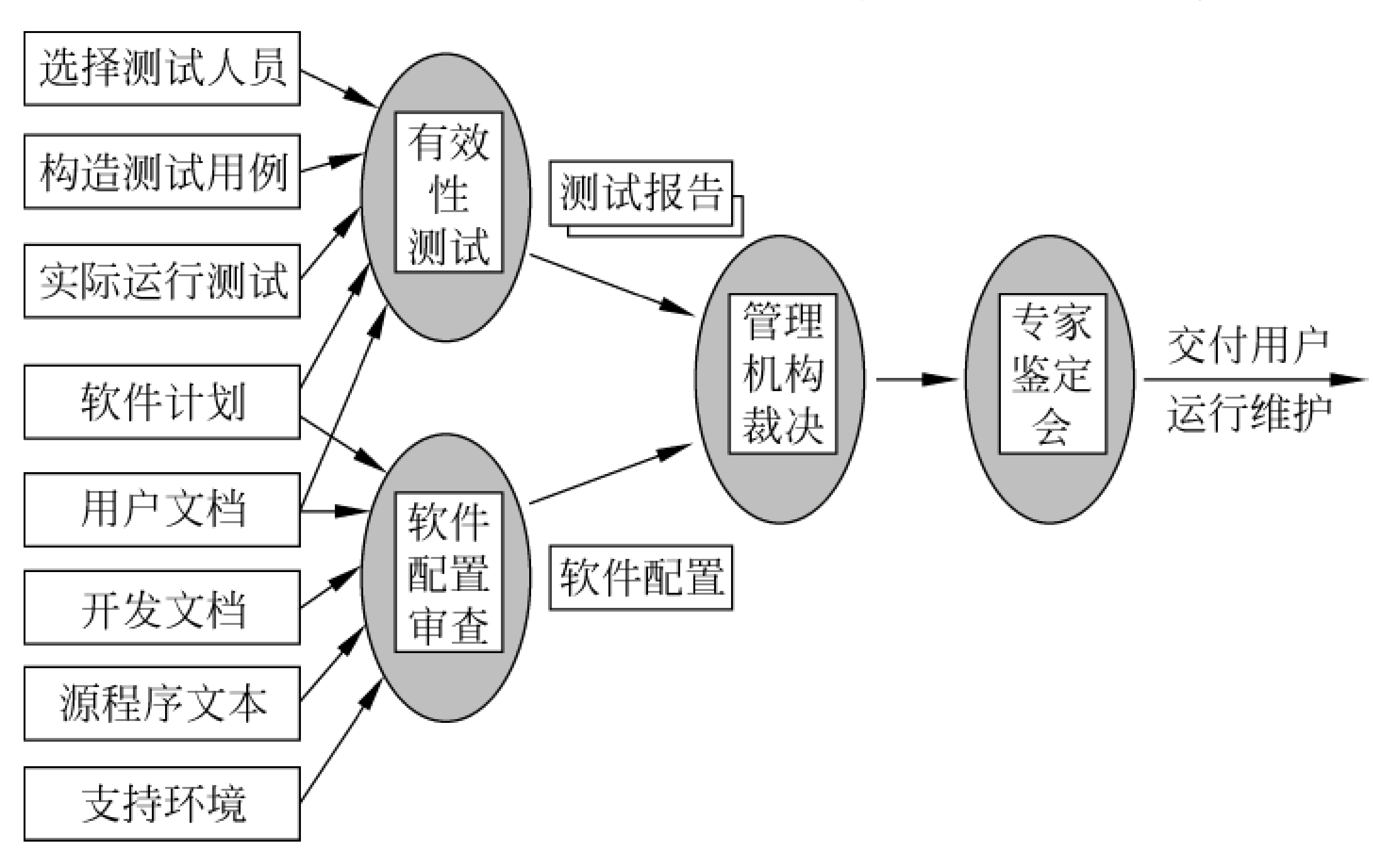


图 2-10 确认测试阶段的工作

确认测试又称为有效性测试。它的任务是验证软件的功能和性能及其特性是否与客户的要求一致。对软件的功能和性能要求在软件需求规格说明中已经明确规定。

2. 确认测试的准则

如何判断被开发的软件是否成功呢？为了确认它的功能、性能以及限制条件是否达到了要求，应该怎样进行测试呢？在需求规格说明书中可能作了原则性规定，但在测试阶段需要更详细、更具体地在测试规格说明书（Test Specification）中作进一步说明。例如，制定测试计划时，要说明确认测试应该测试哪些方面，并给出必要的测试用例。除了考虑功能和性能以外，还需要检验其他方面的要求。例如，可移植性、兼容性、可维护性、人机接口以及开发的文件资料等是否符合要求。经过确认测试，应该为已开发的软件做出结论性评价。有以下两种情况：

- 经过检验的软件功能、性能及其他要求均已满足需求规格说明书的规定，因而可被接受，视为合格的软件；
- 经过检验发现与需求说明书有相当的偏离，得到一个各项缺陷的清单。

对于第二种情况，往往很难在交付期以前把发现的问题纠正过来。这就需要开发部门和客户进行协商，找出解决的办法。

3. 进行确认测试

确认测试是在模拟的环境（可能是就是开发的环境）下，运用黑盒测试的方法，验证所测试件是否满足需求规格说明书列出的需求。为此，需要首先制定测试计划，规定要做测试的种类，还需要制定一组测试步骤，描述具体的测试用例。通过实施预定的测试计划和测试步骤，确定软件的特性是否与需求相符，确保所有的软件功能需求都能得到满足，所有的软件性能需求都能达到，所有的文档都是正确且易于使用。同时，对其他软件需求，例如可移植性、兼容性、自动恢复、可维护性等，也都要进行测试，确认是否满足。

4. 确认测试的结果

在全部软件测试的测试用例运行完后，所有的测试结果可以分为两类：

- 测试结果与预期的结果相符。说明软件的这部分功能或性能特征与需求规格说明书相符合，从而这部分程序被接受。
- 测试结果与预期的结果不符。说明软件的这部分功能或性能特征与需求规格说明不一致，因此要为它提交一份问题报告。

通过与用户的协商，解决所发现的缺陷和错误。确认测试应交付的文档有：确认测试分析报告、最终的用户手册和操作手册、项目开发总结报告。

5. 软件配置审查

软件配置审查是确认测试过程的重要环节。其目的是保证软件配置的所有成分都齐全，各方面的质量都符合要求，具备维护阶段所必需的细节资料并且已经编排好分类的目

录。除了按合同规定的内容和要求,由工人审查软件配置之外,在确认测试的过程,应当严格遵守用户手册和操作手册中规定的使用步骤,以便检查这些文档资料的完整性和正确性。必须仔细记录发现的遗漏和错误,并且适当地补充和改正。

2.2.5 系统测试

1. 系统测试的定义

软件产品离不开运行环境,最终还是要和系统中的其他部分,比如硬件系统、数据信息等集成起来。因此,在投入运行以前要完成系统测试(System Testing),以保证各组成部分不仅能单独地得到检验,而且在系统各部分协调工作的环境下也能正常工作。尽管每一个检验有特定的目标,然而所有的检测工作都要验证系统中每个部分均得到正确的集成,并完成制定的功能。在软件的各类测试中,系统测试是最接近于人们的日常测试实践。它是将已经集成好的软件系统,作为整个计算机系统的一个元素,与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起,在实际运行环境下,对计算机系统进行一系列的组装测试和确认测试。

2. 系统测试的流程

系统测试流程如图 2-11 所示。由于系统测试的目的是验证最终软件系统是否满足产品需求并且遵循系统设计,所以在完成产品需求和系统设计文档之后,系统测试小组就可以提前开始制定测试计划和设计测试用例,不必等到集成测试阶段结束。这样可以提高系统测试的效率。

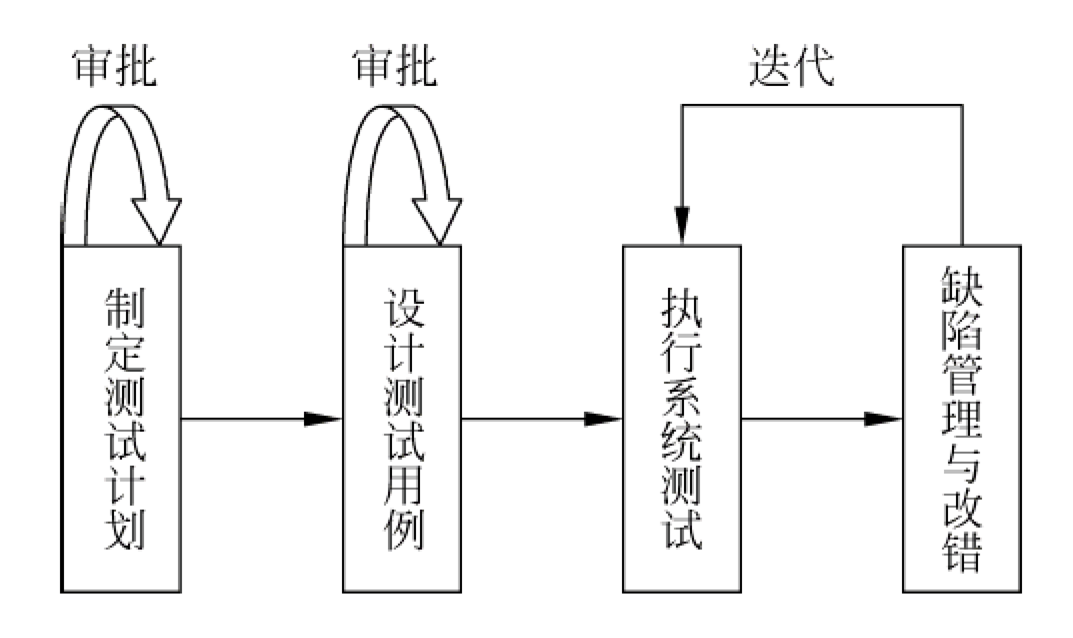


图 2-11 系统测试流程

3. 系统测试的目标

- 确保系统测试的活动是按计划进行的;
- 验证软件产品是否与系统需求用例不相符合或与之矛盾;
- 建立完善的系统测试缺陷记录跟踪库;
- 确保软件系统测试活动及其结果及时通知相关小组和个人。

4. 系统测试的方针

- 为项目指定一个测试工程师负责贯彻和执行系统测试活动；
- 测试组向各事业部总经理/项目经理报告系统测试的执行状况；
- 系统测试活动遵循文档化的标准和过程；
- 向外部用户提供经系统测试验收通过的项目；
- 建立相应项目的缺陷库，用于系统测试阶段项目不同生命周期的缺陷记录和缺陷状态跟踪；
- 定期对系统测试活动及结果进行评估，向各事业部经理/项目办总监/项目经理汇报项目的产品质量信息及数据。

5. 系统测试的设计

为了保证系统测试质量，必须在测试设计阶段就对系统进行严密的测试设计。这就需要在测试设计中，从多方面考虑系统规格的实现情况。通常需要从以下几个层次来进行设计：用户层、应用层、功能层、子系统层、协议/指标层。

- 用户层：主要是面向产品最终的使用操作者的测试。这里重点突出的是站在操作者角度上，测试系统对用户支持的情况，用户界面的规范性、友好性、可操作性，以及数据的安全性。主要包括：用户支持测试、用户界面测试、可维护性测试、安全性测试。
- 应用层：针对产品工程应用或行业应用的测试。重点站在系统应用的角度，模拟实际应用环境，对系统的兼容性、可靠性、性能等进行的测试。主要有：系统性能测试、系统可靠性测试、系统稳定性测试、系统兼容性测试、系统组网测试、系统安装升级测试。
- 功能层：针对产品具体功能实现的测试。主要包括：业务功能的覆盖、业务功能的分解、业务功能的组合、业务功能的冲突。
- 子系统层：针对产品内部结构性能的测试。关注子系统内部的性能，模块间接口的瓶颈。主要内容：单个子系统的性能、子系统间的接口瓶颈、子系统间的相互影响。
- 协议/指标层：针对系统支持的协议、指标的测试。测试内容：协议一致性测试、协议互通测试。

6. 几种常见的系统测试方法

(1) 恢复测试

恢复测试也叫容错测试，用来检查系统的容错能力。通常若计算机系统出现错误，就必须在一定时间内从错误中恢复过来，修正错误并重新启动系统。

恢复测试是通过各种手段，让软件强制性地出错，使其不能正常工作，从而检验系统的恢复能力。对于自动恢复系统，即由系统自身完成恢复工作，则应该检验重新初始化、

检查点、数据恢复和重新启动等机制的正确性。对于人工干预恢复系统，要评估平均修复时间是否在可接受的范围。

（2）安全测试

安全测试的目的在于检查系统对外界非法入侵的防范能力。在安全测试过程中，测试者扮演着非法入侵者，采用各种手段试图突破防线，攻击系统。例如，测试者可以尝试通过外部的手段来破译系统的密码，或者可以有目的地引发系统错误，试图在系统恢复过程中侵入系统等。

系统的安全测试要设置一些测试用例试图突破系统的安全保密防线，用来查找系统的安全保密的漏洞。

系统安全测试的准则是让非法侵入者攻击系统的代价大于保护系统安全的价值。

（3）强度测试

强度测试也称压力测试、负载测试。强度测试是要破坏程序，检测非正常的情况系统的负载能力。强度测试模拟实际情况下的软硬件环境和用户使用过程的系统负荷，长时间或超负荷地运行测试软件来测试系统，以检验系统能力的最高限度，从而了解系统的可靠性、稳定性等。例如，将输入的数据值提高一个或几个数量级来测试输入功能的响应等。

实际上，强度测试就是在一些特定情况下所做的敏感测试。比如数学算法中，在一个有效的数据范围内定义一个极小范围的数据区间，这个数据区间中的数据本应该是合理的，但往往又可能会引发异常的状况或是引起错误的运行，导致程序的不稳定性。敏感测试就是为了发现这种在有效的输入数据区域内可能会引发不稳定性或者引起错误运行的数据集合和组合。

（4）性能测试

性能测试用来测试软件在系统运行时的性能表现，比如运行速度、系统资源占有或响应时间等情况。对于实时系统或嵌入式系统，若只能满足功能需求而不能满足性能需求，是不能被接受的。

性能测试可以在测试过程的任意阶段进行，例如，在单元层，一个独立的模块也可以运用白盒测试方法进行性能评估。但是，只有当一个系统的所有部分都集成后，才能检测此系统的真正性能。

（5）容量测试

容量测试是指在系统正常运行的范围内测定系统能够处理的数据容量，测试系统承受超额数据容量的能力。系统容量必须满足用户需求，如果不能满足实际要求，必须努力改进，寻求解决办法。暂时无法解决的需要在产品说明书中给予说明。

（6）正确性测试

正确性测试是为了检测软件的各项功能是否符合产品规格说明的要求。软件的正确

性与否关系着软件的质量好坏，所以非常重要。

正确性测试的总体思路是设计一些逻辑正确的输入值，检查运行结果是不是期望值。

正确性测试主要有两种方法，一种是枚举法，另一种是边界值法。

对于枚举法，其特点是在测试时应尽量设法减少枚举的次数，从而降低测试的投入成本。次数减少的关键因素就是正确寻找等价区间，因为在等价区间里，只要随意选取一个值测试一次就可以了。

数学定义中，等价区间的概念如下：若(a,b)是命题 $f(x)$的一个等价区间，在(a,b)中任意取 $x1$ 进行测试。如果 $f(x1)$错误，那么 $f(x)$在整个(a,b)区间都将出错；如果 $f(x1)$正确，那么 $f(x)$在整个(a,b)区间都将正确。

枚举法需要凭借直觉和经验来找到等价区间，在程序相当复杂的情况下，枚举测试就显得很有难度。

另一种有效的测试方法就是边界值测试，即采用定义域或者等价区间的边界值进行测试。因为程序设计人员很容易疏忽边界值，程序也最容易在边界值上出问题。例如，测试平方根函数的一段程序，凭直觉输入等价区间应是$(0,1)$和$(1,+\infty)$。可取 $x=0.5$ 以及 $x=0.2$ 进行等价测试，再取 $x=0$ 以及 $x=1$ 进行边界值测试。

(7) 可靠性测试

可靠性测试是从验证的角度出发，检验系统的可靠性是否达到预期的目标，同时给出当前系统可能的可靠性增长情况。可靠性测试需要从用户角度出发，模拟用户实际使用系统的情况，设计出系统的可操作视图。在这个基础上，根据输入空间的属性以及依赖关系导出测试用例，然后在仿真的环境或真实的环境下执行测试用例并记录测试的数据。

对可靠性测试来说，最关键的是测试软件系统的失效间隔时间、失效修复时间、失效数量、失效级别数据等。根据获得的测试数据，应用可靠性模型，可以得到系统的失效率以及可靠性增长趋势。

(8) 兼容性测试

现今，客户对各个开发商和各种软件之间相互兼容、共享数据的能力要求越来越高，所以对于软件兼容性的测试就非常重要。

软件兼容性测试是检测各软件之间能否正常地交互、共享信息，能否正确地和其他软件合作完成数据处理。从而保障软件能够按照客户期望的标准进行交互，多个软件共同完成指定的任务。

交互可以在运行于同一台计算机上的两个程序之间进行，也可以通过因特网，在远距离连接的两个程序间进行。交互也可以简化为在移动存储设备上保存数据，再在其他计算机上运行。

兼容性的测试通常需要解决以下问题：新开发的软件需要与哪种操作系统、Web 浏览器和应用软件保持兼容，如果要测试的软件是一个平台，那么要求应用程序能在其上运

行。应该遵守哪种定义软件之间交互的标准或者规范。软件使用何种数据与其他平台、与新的软件进行交互和共享信息。

兼容性通常有以下几种：

① 向前兼容与向后兼容。向前兼容是指可以使用软件的未来版本，向后兼容是指可以使用软件的以前版本。并非所有的软件都能够向前兼容和向后兼容。

② 不同版本间的兼容。实现测试平台和应用软件多个版本之间能够正常工作是一项困难的任务。例如，现在要测试一个流行的操作系统的新版本，当前的操作系统可能包含上百万程序，新操作系统要求与之百分之百兼容。因为不可能在一个操作系统上测试所有的软件程序，因此需要决定哪些是最重要的，必须要进行的。对于测试新的应用软件程序也一样，需要决定在何种平台上进行测试，与什么样的应用程序一起测试。

③ 标准和规范。适用于软件平台的标准和规范有两个级别：高级标准和低级标准。

高级标准是产品应当普遍遵守的，例如，软件能在何种操作系统上运行？是因特网上的程序吗？它运行于何种浏览器？每一项问题都关系到平台，假若应用程序声明与某个平台兼容，就必须遵守关于该平台的标准和规范。例如，"MS Windows"是微软公司的认证徽标，为了得到这个徽标，软件必须通过独立测试实验室的兼容性测试，其目的就是确保软件在 Windows 操作系统上能平稳可靠地运行。

低级标准是对产品开发细节的描述，从某种意义上说，低级标准比高级标准更加重要。假如创建了一个运行在 Windows 之上的程序，但它与其他 Windows 软件在界面和操作上都有很大的不同，结果是它不会获得"MS Windows"认证徽标。如果是一个图形软件，保存的文件格式却不符合图片文件扩展名的标准，用户就无法在其他程序中查看该文件。软件与标准不兼容，基本上将较快地被淘汰。

同样，在通信协议、编程语言的语法以及用于共享信息的任何形式都必须符合公开的标准与规范。

④ 数据共享兼容。应用程序之间共享数据增强了软件功能。支持并遵守公开的标准，允许用户与其他软件无障碍地传输数据，这个程序就是一个兼容性好的产品。

在 Windows 环境下，剪切、复制和粘贴是程序间常见的一种数据共享方式。在此状况下，传输通过剪贴板的程序来实现。剪贴板设计能存放各种不同的数据类型。Windows 中常见的数据类型包括文本、图片和声音等，这些数据类型可以有各种格式。若对某个程序进行兼容性测试，就要确认其数据能够利用剪贴板与其他程序进行相互复制。其后有大量的代码支持这一兼容特性，其中的测试工作也是一项挑战。另外，通常我们最熟悉的数据共享方式是读写移动外存，如软磁盘、U 盘、移动硬盘等，但文件的数据格式必须符合标准，才能在多台计算机上保持兼容。

(9) Web 网站测试

Web 网站测试是面向因特网 Web 页面的测试。众所周知，因特网网页是由文字、图

形、声音、视频和超级链接等组成的文档。网络客户端用户通过在浏览器中的操作，搜索浏览所需要的信息资源。

针对 Web 网站这一特定类型软件的测试，包含了许多测试技术，如功能测试、压力/负载测试、配置测试、兼容性测试、安全性测试等。黑盒测试、白盒测试、静态测试和动态测试都有可能被采用。

通常 Web 网站测试的内容包含以下方面：

- 功能测试；
- 性能测试；
- 安全性测试；
- 可用性/易用性测试；
- 配置和兼容性测试；
- 数据库测试；
- 代码合法性测试；
- 完成测试。

Web 网站测试将在本书第 9 章中作详细介绍。

2.2.6　验收测试

1. 验收测试的定义

验收测试(Acceptance Testing)是向未来的用户表明系统能够像预定要求的那样工作。通过综合测试之后，软件已完全组装起来，接口方面的错误也已排除，软件测试的最后一步——验收测试即可开始。验收测试是软件产品完成了功能测试和系统测试之后，在产品发布之前所进行的软件测试活动。通过了验收测试，产品正式进入发布阶段。

2. 验收测试的内容

软件验收测试应完成的工作内容如下：要明确验收项目，规定验收测试通过的标准；确定测试方法；决定验收测试的组织机构和可利用的资源；选定测试结果分析方法；指定验收测试计划并进行评审；设计验收测试所用的测试用例；审查验收测试的准备工作；执行验收测试；分析测试结果；做出验收结论，明确通过验收或不通过验收，给出测试结果。

在验收测试计划当中，可能包括的检验方面有以下几种：

(1) 功能测试，如完整的工资计算过程。

(2) 逆向测试，如检验不符合要求数据而引起出错的恢复能力。

(3) 特殊情况，如极限测试、不存在的路径测试。

(4) 文档检查。

(5) 强度检查，如大批量的数据或者最大用户并发使用。

(6) 恢复测试,如硬件故障或用户不良数据引起的一些情况。

(7) 可维护性的评价。

(8) 用户操作测试,如启动、退出系统等。

(9) 用户友好性检验。

(10) 安全测试。

3. 验收测试的标准

实现软件确认要通过一系列黑盒测试。验收测试同样需要制订测试计划和过程,测试计划应规定测试的种类和测试进度,测试过程则定义一些特殊的测试用例,旨在说明软件与需求是否一致。无论是计划还是过程,都应该着重考虑软件是否满足合同规定的所有功能和性能,文档资料是否完整、准确,人机界面和其他方面(例如,可移植性、兼容性、错误恢复能力和可维护性等)是否令用户满意。

验收测试的结果有两种可能,一种是功能和性能指标满足软件需求说明的要求,用户可以接受;另一种是软件不满足软件需求说明的要求,用户无法接受。如果项目进行到这个阶段才发现有严重错误和偏差一般很难在预定的工期内改正,因此必须与用户协商,寻求一个妥善解决问题的方法。

4. 验收测试的常用策略

选择的验收测试的策略通常建立在合同需求、组织和公司标准以及应用领域的基础上。实施验收测试的常用策略有三种,下面简要介绍。

(1) 正式验收测试

正式验收测试是一项管理严格的过程,它通常是系统测试的延续。计划和设计这些测试的周密和详细程度不亚于系统测试。选择的测试用例应该是系统测试中所执行测试用例的子集。不要偏离所选择的测试用例方向,这一点很重要。在很多组织中,正式验收测试是完全自动执行的。对于系统测试,活动和工件是一样的。在某些组织中,开发组织(或其独立的测试小组)与最终用户组织的代表一起执行验收测试。在其他组织中,验收测试则完全由最终用户组织执行,或者由最终用户组织选择人员组成一个客观公正的小组来执行。

正式验收测试的优点是要测试的功能和特性都是明确的;测试的细节是已知的并且可以对其进行评测;测试可以自动执行,支持回归测试;可以对测试过程进行评测和监测;可接受性标准是已知的。

当然,正式验收测试也有缺点,主要有:测试要求大量的资源和计划,而且这些测试可能是系统测试的再次实施,也可能无法发现软件中由于主观原因造成的缺陷,这是因为只查找了预期要发现的缺陷。

(2) 非正式验收或 Alpha 测试

在非正式验收测试中,执行测试过程的限定不像正式验收测试中那样严格。在此测

试中,确定并记录要研究的功能和业务任务,但没有可以遵循的特定测试用例。测试内容由各测试员决定。这种验收测试方法不像正式验收测试那样组织有序,而且更为主观。大多数情况下,非正式验收测试是由最终用户组织执行的。

非正式验收测试的优点是:要测试的功能和特性都是已知的;可以对测试过程进行评审和监测;可接受性标准是已知的;非正式验收测试和正式验收测试相比,可以发现更多由于主观原因造成的缺陷。

非正式验收测试的缺点包括:要求资源、计划和管理资源;无法控制所使用的测试用例;最终用户可能沿用系统工作的方式,并可能无法发现缺陷;最终用户可能更专注于比较新系统与遗留系统,而不是专注于查找缺陷;用于验收测试的资源不受项目的控制,并且可能受到压缩。

(3) Beta 测试

与以上两种验收测试策略相比,Beta 测试需要的控制是最少的。在 Beta 测试中,采用的细节多少、数据和方法完全由各测试员决定。各测试员负责创建自己的环境、选择数据,并决定要研究的功能、特性或任务。各测试员负责确定自己对于系统当前状态的接受标准。Beta 测试由最终用户实施,通常开发组织对其管理很少或不进行管理。Beta 测试是所有验收测试策略中最主观的。

Beta 测试形式的优点是:测试由最终用户实施;大量的潜在测试资源;提高客户对参与人员的满意程度;与正式或非正式验收测试相比,可以发现更多由于主观原因造成的缺陷。

Beta 测试的缺点包括:未对所有功能/特性测试;测试流程难以评测;最终用户可能沿用系统工作的方式,并可能没有发现或没有报告缺陷;最终用户可能更专注于比较新的系统与遗留系统,而不是专注于查找缺陷;用于验收测试的资源不受项目的控制,并且可能受到压缩;可接受性标准是未知的;需要更多辅助性资源来管理 Beta 测试员。

5. 验收测试的过程

验收测试的工作流程如图 2-12 所示,主要步骤如下:

- 验收测试的项目洽谈。
- 验收测试合同。
- 提交测试样品及机关资料。

(1) 软件需求分析:要分析测试样品及其相关资料,要了解软件功能和性能要求、软硬件环境要求等,并特别要了解软件的质量要求和验收要求。综合分析产品是否达到验收测试状态,未达到验收测试状态,要返回提交测试样品及相关资料;达到测试状态,则可向下执行。

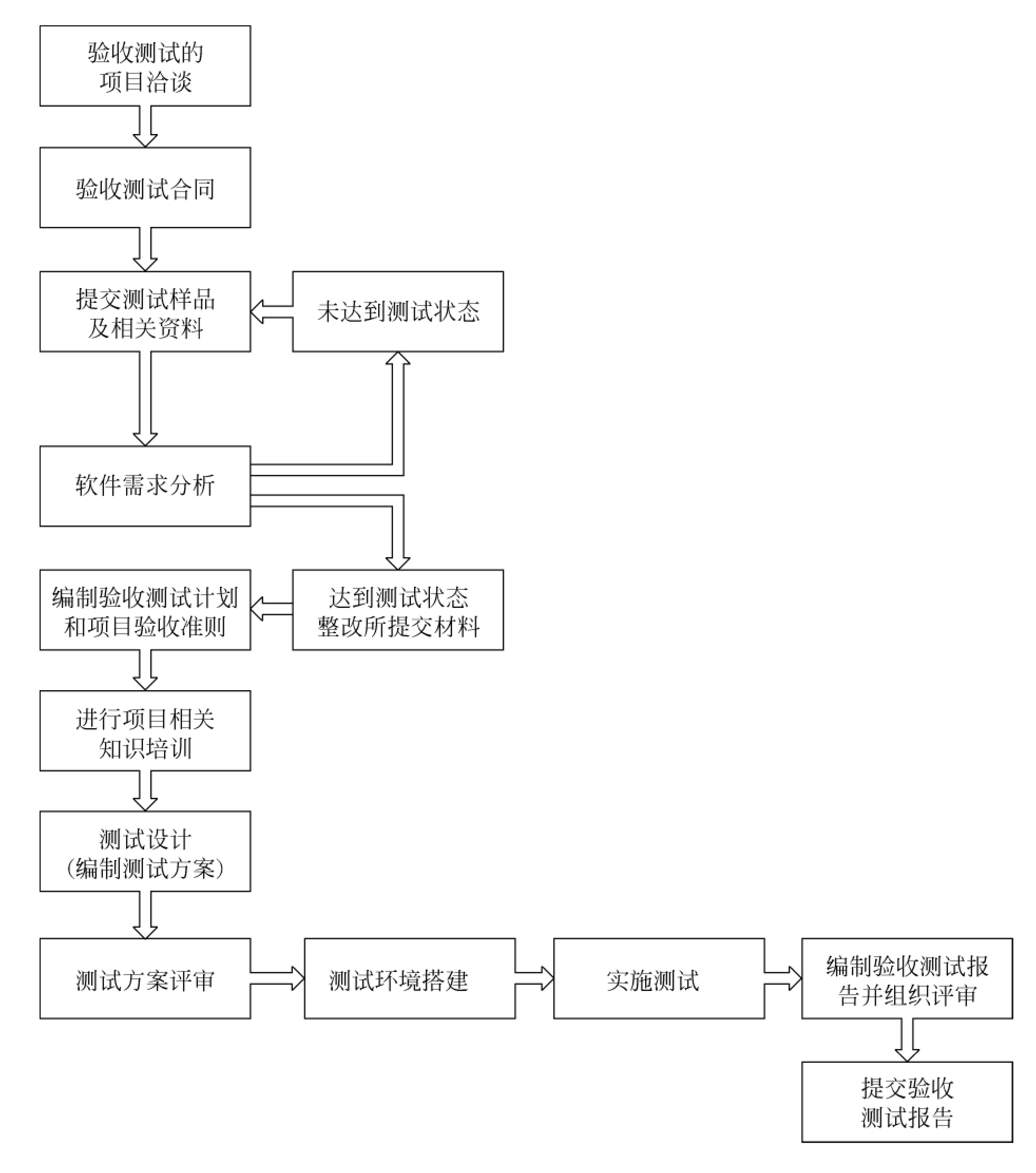


图 2-12 验收测试的工作流程

(2) 编制《验收测试计划》和《项目验收准则》：测试计划在需求分析阶段建立，根据软件需求和验收要求编制测试计划，制定需测试的测试项，制定测试策略及验收通过准则，并经过客户参与相关计划的评审。

(3) 进行项目相关知识培训。

(4) 测试设计和测试用例设计：根据《验收测试计划》和《项目验收准则》编制测试用例和相关方案。

(5) 测试方案评审：评审测试实施方案和相关测试用例。

（6）测试环境搭建：建立测试的硬件环境、软件环境等（可在委托客户提供的环境中进行测试）。

（7）实施测试：进行验收测试并记录测试结果。

（8）编制验收测试报告并组织评审：根据验收通过准则分析测试结果，做出验收是否通过及测试评价。

（9）提交验收测试报告：根据测试结果编制缺陷报告和验收测试报告，并提交给客户。

6. 验收测试的总体思路

用户验收测试是软件开发结束后，用户对软件产品投入实际应用以前进行的最后一次质量检验活动。它要回答开发的软件产品是否符合预期的各项要求，以及用户能否接受的问题。由于它不只是检验软件某个方面的质量，而是要进行全面的质量检验，并且要决定软件是否合格，因此验收测试是一项严格的正式测试活动。

用户验收测试可以分为两个大的部分：软件配置审核和可执行程序测试。其大致顺序可分为：文档审核、源代码审核、配置脚本审核、测试程序或脚本审核、可执行程序测试。

（1）软件配置审核

对于一个外包的软件项目而言，软件承包方通常要提供如下相关的软件配置内容：

- 可执行程序、源程序、配置脚本、测试程序或脚本；
- 主要的开发类文档：《需求分析说明书》、《概要设计说明书》、《详细设计说明书》、《数据库设计说明书》、《测试计划》、《测试报告》、《程序维护手册》、《程序员开发手册》、《用户操作手册》、《项目总结报告》；
- 主要的管理类文档：《项目计划书》、《质量控制计划》、《配置管理计划》、《用户培训计划》、《质量总结报告》、《评审报告》、《会议记录》、《开发进度月报》。

不论项目大小，都必须具备上述的文档内容，只是可以根据实际情况进行重新组织。审核要达到的基本目标是：根据共同制定的审核表，尽可能地发现被审核内容中存在的问题，并最终得到解决。在根据相应的审核表进行文档审核和源代码审核时，还要注意文档与源代码的一致性。

在实际的验收测试执行过程中，常常会发现文档审核是最难的工作，一方面由于市场需求等方面的压力使这项工作常常被弱化或推迟，造成持续时间变长，加大文档审核的难度；另一方面，文档审核中不易把握的地方非常多，每个项目都有一些特别的地方，而且也很难找到可用的参考资料。

（2）可执行程序的测试

当文档审核、源代码审核、配置脚本审核、测试程序或脚本审核都顺利完成后，就可以进行验收测试的最后一个步骤——可执行程序的测试，它包括功能、性能等方面的测试，

每种测试都包括目标、启动标准、活动、完成标准和度量五部分。

在真正进行用户验收测试之前一般已经完成了以下工作(也可以根据实际情况选取或增加):

- 软件开发已经完成,并全部解决了已知的软件缺陷;
- 验收测试计划已经过评审并批准,并且置于文档控制之下;
- 对软件需求说明书的审查已经完成;
- 对概要设计、详细设计的审查已经完成;
- 对所有关键模块的代码审查已经完成;
- 对单元、集成、系统测试计划和报告的审查已经完成;
- 所有的测试脚本已完成,并至少执行过一次,且通过评审;
- 使用配置管理工具且代码置于配置控制之下;
- 软件问题处理流程已经就绪;
- 已经制定、评审并批准验收测试完成标准。

具体的测试内容通常包括:

- 安装(升级);
- 启动与关机;
- 功能测试(正例、重要算法、边界、时序、反例、错误处理);
- 性能测试(正常的负载、容量变化);
- 压力测试(临界的负载、容量变化);
- 配置测试;
- 平台测试;
- 安全性测试;
- 恢复测试(在出现掉电、硬件故障或切换、网络故障等情况时,系统是否能够正常运行);
- 可靠性测试;
- 性能测试和压力测试。

一般情况下性能测试和压力测试二者在一起进行,通常还需要辅助工具的支持。在进行性能测试和压力测试时,测试范围必须限定在那些使用频度高和时间要求苛刻的软件功能子集中。由于开发方已经事先进行过性能测试和压力测试,因此可以直接使用开发方的辅助工具。也可以通过购买或自己开发来获得辅助工具。如果执行了所有的测试案例、测试程序或脚本,用户验收测试中发现的所有软件问题都已解决,而且所有的软件配置均已更新和审核,可以反映出软件在用户验收测试中所发生的变化,用户验收测试就完成了。

2.3　静态测试与动态测试

根据程序是否运行可以把软件测试方法分为静态测试(Static Testing)和动态测试(Dynamic Testing)两大类。图 2-13 是静态测试与动态测试的比喻图。

踩油门
看车漆　　静态测试
打开前盖检查
发动汽车
听听发动机声音　动态测试
上路行使

图 2-13　静态测试与动态测试的比喻图

2.3.1　静态测试

静态方法的主要特征是在用计算机测试源程序时，计算机并不真正运行被测试的程序，只对被测程序进行特性分析。因此，静态方法常称为“分析”，静态分析是对被测程序进行特性分析的一些方法的总称。所谓静态分析，就是不需要执行所测试的程序，而只是通过扫描程序正文，对程序的数据流和控制流等信息进行分析，找出系统的缺陷，得出测试报告。

为什么要进行静态分析呢？首先，一个软件产品可能实现了所要求的功能，但如果它的内部结构组织得很复杂，很混乱，代码的编写也没有规范，这时软件中往往会隐藏一些不易被察觉的错误。其次，即使这个软件基本满足了用户目前的要求，但到了日后对该产品进行维护升级工作的时候，会发现维护工作相当困难。所以，如果能对软件进行科学、细致的静态分析，使系统的设计符合模块化、结构化、面向对象的要求，使开发人员编写的代码符合规定的编码规范，就能够避免软件中大部分的错误，同时为日后的维护工作节约大量的人力、物力。这就是对软件进行静态分析的价值所在。

静态测试包括代码检查、静态结构分析、代码质量度量等。它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工具自动进行。

通常在静态测试阶段进行以下一些测试活动：

- 检查算法的逻辑正确性，确定算法是否实现了所要求的功能。
- 检查模块接口的正确性，确定形参的个数、数据类型、顺序是否正确，确定返回值类型及返回值的正确性。
- 检查输入参数是否有合法性检查。如果没有合法性检查，则应确定该参数是否不需要合法性检查，否则应加上参数的合法性检查。

- 检查调用其他模块的接口是否正确，检查实参类型、实参个数是否正确，返回值是否正确。若被调用模块出现异常或错误，程序是否有适当的出错处理代码。
- 检查是否设置了适当的出错处理，以便在程序出错时，能对出错部分进行重新安排，保证其逻辑的正确性。
- 检查表达式、语句是否正确，是否含有二义性。例如，下列表达式或运算符的优先级：<=、=、>=、&&、||、++、--等。
- 检查常量或全局变量使用是否正确。
- 检查标识符的使用是否规范、一致，变量命名是否能够望名知义、简洁、规范和易记。
- 检查程序风格的一致性、规范性，代码是否符合行业规范，是否所有模块的代码风格一致、规范。
- 检查代码是否可以优化，算法效率是否最高。
- 检查代码注释是否完整，是否正确反映了代码的功能，并查找错误的注释。

静态分析的差错分析功能是编译程序所不能替代的。编译系统虽然能发现某些程序错误，但这些错误远非软件中存在的大部分错误。目前，已经开发了一些静态分析系统作为软件静态测试的工具，静态分析已被当做一种自动化的代码校验方法。

静态测试可以完成的工作如下：

(1) 可以发现如下的程序缺陷

- 错用了局部变量和全局变量；
- 不匹配的参数；
- 未定义的变量；
- 不适当的循环嵌套或分支嵌套；
- 无终止的死循环；
- 不允许的递归；
- 调用不存在的子程序；
- 遗漏了标号或代码。

(2) 找出如下问题的根源

- 未使用过的变量；
- 不会执行到的代码；
- 从未引用过的标号；
- 潜在的死循环。

(3) 提供程序缺陷的如下间接信息

- 标识符的使用方式；
- 过程的调用层次；

- 所用变量和常量的交叉应用表；
- 是否违背编码规则。

(4) 为进一步查错做准备

(5) 选择测试用例

(6) 进行符号测试

实践表明，使用静态测试可发现大约 1/3～2/3 的逻辑设计和编码错误。但代码中仍会隐藏许多故障无法通过静态测试来发现，因此必须通过动态测试进行详细分析。

2.3.2　动态测试

动态方法是通过源程序运行时所体现出来的特征，来进行执行跟踪、时间分析以及测试覆盖等方面的测试。动态测试是真正运行被测程序，在执行过程中，通过输入有效的测试用例，对其输入与输出的对应关系进行分析，以达到检测的目的。

动态测试方法的基本步骤：

(1) 选取定义域的有效值，或选取定义域外的无效值；

(2) 对已选取值决定预期的结果；

(3) 用选取值执行程序；

(4) 执行结果与预期的结果相比，不吻合则证明程序有错。

不同的测试方法各自的目标和侧重点不一样，在实际工作中要将静态测试和动态测试结合起来，以达到更加完美的效果。

在动态测试运行中，又可有基于程序结构的白盒测试(或称为覆盖测试)和基于功能的黑盒测试。

2.4　黑盒测试与白盒测试

测试用例的设计是测试过程的一个关键步骤，按照测试用例的不同出发点，可以分为黑盒测试与白盒测试。一般来讲，在进行单元测试时采用白盒测试，而其余测试采用黑盒测试。

2.4.1　黑盒测试

黑盒测试(Black-Box Testing)又称为功能测试、数据驱动测试和基于规格说明的测试，是一种从用户观点出发的测试。

黑盒测试的基本观点是：任何程序都可以看作是从输入定义域映射到输出值域的函数过程，被测程序被认为是一个打不开的黑盒子，黑盒中的内容(实现过程)完全不知道，只明确要做到什么。黑盒测试作为软件功能的测试手段，是重要的测试方法。它主要根据规格说明设计测试用例，并不涉及程序内部结构和内部特性，只依靠被测程序输入和输出之间的关系或程序的功能设计测试用例。

黑盒测试有两个显著的特点：

(1) 黑盒测试不考虑软件的具体实现过程，当软件实现的过程发生变化时，测试用例仍然可以使用；

(2) 黑盒测试用例的设计可以和软件实现同时进行，这样能够压缩总的开发时间。

黑盒测试不仅能够找到大多数其他测试方法无法发现的错误，而且一些外购软件、参数化软件包以及某些自动生成的软件，由于无法得到源程序，在一些情况下只能选择黑盒测试。

但是任何一个软件作为一个系统都是有层次的，在软件的总体功能之下可能具有若干个层次的功能，而且软件开发一般总是将原始问题换算成计算机能够处理的形式作为开始，接下来进行一系列变换，最后得到程序编程。在这一系列变换的过程之中，每一步都得到不同形式的中间成果，再生成相应功能。因此，测试人员常常面临的一个实际问题是在哪个层次上进行测试。假如是在高一层次上进行的测试，就可能忽略一些细节，测试可能是不完全的和不够充分的；假如是在较低一层次上进行的测试，则有可能忽略各功能存在的相互作用和相互依赖的关系。

如果想用黑盒测试发现程序中所有的错误，就必须输入数据的所有可能的值来检查程序是否都能够产生正确的结果。但这显然是做不到的。一方面，输入和输出结果是否正确本身无法全部事先知道；另一方面，要做到穷举所有可能的输入实际上很困难。通常黑盒测试的测试数据是根据规格说明书来决定的，但实际上，也比较难以保证规格说明书是不是完全正确，可能也存在着问题。例如，如果规格说明书中规定了多余的功能，或是遗漏了某些功能，采用黑盒测试是无法发现这些问题的。

黑盒测试的具体技术方法主要包括边界值分析法、等价类划分法、比较测试法、因果图法、决策表法等。

黑盒测试属于穷举输入测试方法，只有把所有可能的输入都作为测试情况来使用，才能以这种方法查出程序中所有的错误。

软件测试仅局限于功能测试是不够的，因此不仅要进行黑盒测试，还需要花费很大的精力进行逻辑(结构)测试，即白盒测试。

2.4.2 白盒测试

白盒测试(White-Box Testing)也称作结构测试或逻辑驱动测试，它是知道产品的内部工作过程，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行。按照程序内部的结构测试程序，检验程序中的每条通路是否都能按预定要求正确工作，而不考虑它的功能。白盒测试的主要方法有逻辑覆盖、基本路径测试等，主要用于软件验证。

白盒测试是全面了解程序内部逻辑结构、对所有逻辑路径进行测试。白盒测试是穷举路径测试。在使用这一方案时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。贯穿程序的独立路径数是天文数字。但即使每条路径都测试了仍然可能有错误。第一，穷举路径测试决不能查出程序是否违反了设计规范，即程序本身就是

个错误的程序。第二，穷举路径测试不可能查出程序中因遗漏路径而出错。第三，穷举路径测试可能发现不了一些与数据相关的错误。

白盒测试的测试规划是基于产品的内部结构来进行测试，检查内部操作是否按规定进行，软件的各个部分功能是否得到充分利用。白盒测试又称结构测试或基于程序的测试，即逻辑测试。白盒测试将被测程序看作一个打开的盒子，测试者能够看到被测源程序，可以分析被测程序的内部结构，此时测试的焦点集中在根据其内部结构设计测试用例。

既然白盒测试是根据被测程序的内部结构来设计测试用例的一类测试，也许有人会认为，只要保证程序中所有的路径都执行一次，全面的白盒测试将产生“百分之百正确的程序”。这实际上是不可能的，即便是一个非常小的控制流程，进行穷举测试所需要的时间都是一个巨大的数字。

因此，白盒测试要求对某些程序的结构特性做到一定程度的覆盖，或者说这种测试是“基于覆盖率的测试”。测试人员可以严格定义要测试的确切内容，明确要达到的测试覆盖率，减少测试的过分和盲目并以此为目标，引导测试者朝着提高覆盖率的方向努力，找出那些可能已被忽视的程序错误。

通常的程序结构覆盖有：

- 语句覆盖；
- 判断覆盖；
- 条件覆盖；
- 判断/条件覆盖；
- 条件组合覆盖；
- 路径覆盖。

语句覆盖是最常见也是最弱的逻辑覆盖准则，它要求设计若干个测试用例，使被测程序的每个语句都至少被执行一次。判定覆盖或分支覆盖则要求设计若干个测试用例，使被测程序的每个判定的真、假分支都至少被执行一次。但判定含有多个条件时，可以要求设计若干个测试用例，使被测程序的每个条件的真、假分支都至少被执行一次，即条件覆盖。在考虑对程序路径进行全面检验时，即可使用条件覆盖准则。

虽然结构测试提供了评价测试的逻辑覆盖准则，但结构测试是不完全的。如果程序结构本身存在问题，比如程序逻辑错或者遗漏了规格说明书中已规定的功能，那么，无论哪种结构测试，即使其覆盖率达到了百分之百，也是检查不出来的。因此，提高结构测试的覆盖率，可以增强对被测软件的信度，但并不能做到万无一失。

2.4.3 黑盒测试与白盒测试的对比

黑盒测试法和白盒测试法是从完全不同的起点出发，并且两个出发点在某种程度上是完全不同的，这反映了测试思路的两个方面。两类方法在软件测试实践过程中均被证明是有效和实用的方法。

经验表明，在进行单元测试时通常采用白盒测试法；而在集成测试、确认测试或系统测试时常采用黑盒测试法。

黑盒测试是以用户的观点，从输入数据与输出数据的对应关系，也就是根据程序外部特性进行的测试，而不考虑内部结构及工作情况；黑盒测试技术注重于软件的信息域（范围），通过划分程序的输入和输出域来确定测试用例；若外部特性本身存在问题或规格说明的规定有误，则应用黑盒测试方法是不能发现问题的。反之，白盒测试只根据程序的内部结构进行测试；测试用例的设计要保证测试时程序的所有语句至少执行一次，而且要检查所有的逻辑条件；如果程序的结构本身有问题，比如说程序逻辑有错误或者有遗漏，那也是无法发现的。

小　　结

软件测试的复杂性和经济性说明测试过程需要运用多种策略，针对不同的被测试程序状况选用不同的测试方法。

通常，对于一个大型的软件系统，测试流程由多个必经阶段组成：单元测试、集成测试、确认测试、系统测试和验收测试。

单元测试是在软件测试过程的最基础级别的测试活动，目的是检测程序中的模块无软件故障存在。

集成测试是把通过单元测试的各模块边组装边测试，来检测与程序接口方面的故障。

确认测试是按照软件需求规格说明来验证软件产品是否满足需求规格的要求。

系统测试是对系统中各个组成部分进行综合测试。

验收测试是验收软件产品是否符合预定的各项要求，是否让用户满意。

根据程序是否运行，可以把软件测试方法分为静态测试和动态测试两大类。根据测试步骤的不同出发点，可以分为黑盒测试与白盒测试，二者比较见表 2-1。这两组方法可以进行某种形式的组合，来满足测试要求。

表 2-1　黑盒测试和白盒测试比较

性能 \ 测试法	黑盒测试	白盒测试
优点	① 适用于各个测试阶段 ② 从产品功能角度进行测试 ③ 容易入手生成测试数据	① 可构成测试数据使特定程序部分得到测试 ② 有一定充分性度量手段 ③ 可获较多工具支持
缺点	① 某些代码得不到测试 ② 如果规则说明有误，无法发现 ③ 不易进行充分行测试	① 不易生成测试数据 ② 无法对未实现规格说明的部分进行测试 ③ 工作量大，通常只用于单元测试，有应用局限性
性质	一种确认技术，目的是确认“设计的系统是否正确”	一种验证技术，目的是验证“系统的设计是否正确”

习　　题

1. 简述软件测试的复杂性。
2. 对软件的经济性进行总结和分析。
3. 阐述软件测试的充分性准则。
4. 如何描述测试流程整体框架。
5. 简述单元测试的目标。
6. 解释驱动模块和桩模块的概念。
7. 简述集成测试的层次划分。
8. 归纳确认测试阶段的工作。
9. 简述系统测试的流程。
10. 归纳验收测试常用的策略。
11. 简述验收测试的流程。
12. 简述静态测试和动态测试的区别。
13. 比较并阐述黑盒测试和白盒测试的优缺点。

第 3 章

黑盒测试及其实例设计

本章概述

黑盒测试是软件测试技术中最基本的方法之一，在各类测试中都有广泛的应用。本章将介绍黑盒测试的基本概念与基本方法，并重点介绍应用较为广泛的几种测试方法：等价类划分法、边界值分析法、决策表法和因果图法，并通过典型实例详细介绍实际测试技术的基本运用。

3.1 黑盒测试概述

黑盒测试又称为功能测试或数据驱动测试，是从用户观点出发，主要以软件规格说明书为依据，对程序功能和程序接口进行的测试。

在黑盒测试中，测试条件主要是基于程序或者系统的功能。例如，测试人员需要掌握有关输入数据的信息并观察输出数据，但是测试人员并不知道程序到底是如何工作的，这就好比一个人虽然会开车，但这个人并不知道汽车的内部工作方式。在这里，运行一个程序并不需要理解其内部结构，测试人员只是根据产品应该实现的实际功能和已经定义好的产品规格，来验证产品所应该具有的功能是否实现，每个功能是否都能正常使用，是否满足用户的要求。在测试时，测试人员将整个被测试的程序看成一个黑盒子，在完全不考虑程序或者系统的内部结构和内部特性的情况下，检查程序的功能能否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出结果。

黑盒测试是以用户的观点，从输入数据与输出数据的对应关系出发进行测试的，它不涉及程序的内部结构。很明显，如果外部特性本身有问题或规格说明书的规定有误，用黑盒测试方法是发现不了的。黑盒测试方法着重测试软件的功能需求，是在程序接口上进行测试，主要是为了发现以下错误：

（1）是否有不正确的功能，是否有遗漏的功能；

（2）在接口上，是否能够正确地接收输入数据并产生正确的输出结果；

(3) 是否有数据结构错误或外部信息访问错误；

(4) 性能上是否能够满足要求；

(5) 是否有程序初始化和终止方面的错误。

黑盒测试有两种基本方法，即通过测试和失败测试。

在进行通过测试时，实际上是确认软件能做什么，而不会去考验其能力如何，软件测试人员只是运用最简单、最直观的测试案例。在设计和执行测试案例时，总是先要进行通过测试，验证软件的基本功能是否都已实现。

在确信了软件正确运行之后，就可以采取各种手段通过搞垮软件来找出缺陷。纯粹为了破坏软件而设计和执行的测试案例，被称为失败测试或迫使出错测试。

黑盒测试的一个主要的好处就是它直接针对程序或者系统的目标，容易理解，比较直观。但是黑盒测试属于穷举输入测试方法，这就要求每一个可能的输入或者输入的组合都被测试到，才能查出程序中所有的错误，这通常要受到较大的限制。假设有一个程序要求有两个输入数据 x 和 y 及一个输出数据 z，在字长为 32 位的计算机上运行。若 x、y 取整数，按黑盒测试方法进行穷举测试，则测试数据的最大可能数目为：$2^{32}\times 2^{32}=2^{64}$。如果测试一组数据需要 1 毫秒，一天工作 24 小时，一年工作 365 天，那么完成所有测试需 5 亿年。可见，要进行穷举输入是不可能的。

黑盒测试的各种方法中，应用较为广泛的测试方法有以下几种：等价类划分法、边界值分析法、决策表法和因果图法。这些方法是比较实用的，在项目中具体采用什么方法，在设计具体的测试方案时自然要针对开发项目的特点对设计方法进行适当的选择。

3.2　等价类划分法

3.2.1　等价类划分法的概念

等价类划分法是黑盒测试用例设计中一种常用的设计方法，它将不能穷举的测试过程进行合理分类，从而保证设计出来的测试用例具有完整性和代表性。

等价类划分法是把所有可能的输入数据，即程序的输入域划分成若干部分(子集)，然后从每一个子集中选取少数具有代表性的数据作为测试用例。所谓等价类是指输入域的某个子集合，所有等价类的并集就是整个输入域。在等价类中，各个输入数据对于揭露程序中的错误都是等效的，它们具有等价特性。因此，测试某个等价类的代表值就是等价于对这一类中其他值的测试。也就是说，如果某一类中的一个例子发现了错误，这一等价类中的其他例子也能发现同样的错误；反之，如果某一类中的一个例子没有发现错误，则这一类中的其他例子也不会查出错误。

软件不能只接收合理有效的数据，也要具有处理异常数据的功能，这样的测试才能确保软件具有更高的可靠性。因此，在划分等价类的过程中，不但要考虑有效等价类划分，

同时也要考虑无效等价类划分。

有效等价类是指对软件规格说明来说，合理、有意义的输入数据所构成的集合。利用有效等价类可以检验程序是否满足规格说明所规定的功能和性能。

无效等价类则和有效等价类相反，即不满足程序输入要求或者无效的输入数据所构成的集合。利用无效等价类可以检验程序异常情况的处理。

使用等价类划分法设计测试用例，首先必须在分析需求规格说明的基础上划分等价类，然后列出等价类表。

以下是划分等价类的几个原则：

（1）如果规定了输入条件的取值范围或者个数，则可以确定一个有效等价类和两个无效等价类。例如，程序要求输入的数值是从 10 到 20 之间的整数，则有效等价类为“大于等于 10 而小于等于 20 的整数”，两个无效等价类为“小于 10 的整数”和“大于 20 的整数”。

（2）如果规定了输入值的集合，则可以确定一个有效等价类和一个无效等价类。例如，程序要进行平方根函数的运算，则“大于等于 0 的数”为有效等价类，“小于 0 的数”为无效等价类。

（3）如果规定了输入数据的一组值，并且程序要对每一个输入值分别进行处理，则可为每一个值确定一个有效等价类，此外根据这组值确定一个无效等价类，即所有不允许的输入值的集合。例如，程序规定某个输入条件 x 的取值只能为集合$\{1,3,5,7\}$中的某一个，则有效等价类为 $x=1$，$x=3$，$x=5$，$x=7$，程序对这 4 个数值分别进行处理；无效等价类则为 $x\neq1,3,5,7$ 的值的集合。

（4）如果规定了输入数据必须遵守的规则，则可以确定一个有效等价类和若干个无效等价类。例如，程序中某个输入条件规定必须为 4 位数字，则可划分一个有效等价类为输入数据为 4 位数字，3 个无效等价类分别为输入数据中含有非数字字符、输入数据少于 4 位数字、输入数据多于 4 位数字。

（5）如果已知的等价类中各个元素在程序中的处理方式不同，则应将该等价类进一步划分成更小的等价类。

在确立了等价类之后，建立等价类表，列出所有划分出的等价类，格式如表 3-1 所示。

表 3-1　等价类表

输入条件	有效等价类	无效等价类
…	…	…
…	…	…

再根据已列出的等价类表，按以下步骤确定测试用例：

（1）为每一个等价类规定一个惟一的编号；

（2）设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这

个过程，直至所有的有效等价类均被测试用例所覆盖；

(3) 设计一个新的测试用例，使其仅覆盖一个无效等价类，重复这个过程，直至所有的无效等价类均被测试用例所覆盖。

3.2.2　常见等价类划分形式

针对是否对无效数据进行测试，可以将等价类测试分为标准等价类测试、健壮等价类测试及对等区间划分。

1. 标准等价类测试

标准等价类测试不考虑无效数据值，测试用例使用每个等价类中的一个值。通常，标准等价类测试用例的数量和最大等价类中元素的数目相等。

以三角形问题为例，要求输入三个整数 a、b、c，分别作为三角形的三条边，取值范围为 1～100，判断由三条边构成的三角形类型为等边三角形、等腰三角形、一般三角形以及不构成三角形。在多数情况下，是从输入域划分等价类，但对于三角形问题，从输出域来定义等价类是最简单的划分方法。

因此，利用这些信息可以确定下列值域等价类：

$R1=\{\langle a,b,c\rangle$：边为 a,b,c 的等边三角形$\}$

$R2=\{\langle a,b,c\rangle$：边为 a,b,c 的等腰三角形$\}$

$R3=\{\langle a,b,c\rangle$：边为 a,b,c 的一般三角形$\}$

$R4=\{\langle a,b,c\rangle$：边为 a,b,c 不能构成三角形$\}$

4 个标准等价类测试用例如表 3-2 所示。

表 3-2　三角形问题的标准等价类测试用例

测试用例	a	b	c	预期输出
TC1	20	20	20	等边三角形
TC2	20	20	10	等腰三角形
TC3	5	8	10	一般三角形
TC4	3	6	10	不构成三角形

2. 健壮等价类测试

健壮等价类测试主要的出发点是考虑了无效等价类。

对有效输入，测试用例从每个有效等价类中取一个值；对无效输入，一个测试用例有一个无效值，其他值均取有效值。

健壮等价类测试存在两个问题：

(1) 需要花费精力定义无效测试用例的期望输出；

(2) 对强类型的语言没有必要考虑无效的输入。

对于上述三角形问题，取 a,b,c 的无效值产生了 7 个健壮等价类测试用例，如表 3-3 所示。

表 3-3 三角形问题的健壮等价类测试用例

测试用例	a	b	c	预 期 输 出
TC1	3	4	5	一般三角形
TC2	−1	4	4	a 值超出定义域范围
TC3	4	−1	4	b 值超出定义域范围
TC4	4	4	−1	c 值超出定义域范围
TC5	101	4	4	a 值超出定义域范围
TC6	4	101	4	b 值超出定义域范围
TC7	4	4	101	c 值超出定义域范围

3. 对等区间划分

对等区间划分是测试用例设计的非常规形式化的方法。它将被测对象的输入/输出划分成一些区间，被测软件对一个特定区间的任何值都是等价的。形成测试区间的数据不只是函数/过程的参数，也可以是程序可以访问的全局变量、系统资源等，这些变量或资源可以是以时间形式存在的数据，或以状态形式存在的输入/输出序列。

对等区间划分假定位于单个区间的所有值对测试都是对等的，应为每个区间的一个值设计一个测试用例。

举例说明如下：

平方根函数要求当输入值为 0 或大于 0 时，返回输入数的平方根；当输入值小于 0 时，显示错误信息“平方根错误，输入值小于 0”，并返回 0。

考虑平方根函数的测试用例区间，可以划分出两个输入区间和两个输出区间，如表 3-4 所示。

表 3-4 区间划分

输入区间		输出区间	
i	<0	A	>=0
ii	>=0	B	Error

通过分析，可以用两个测试用例来测试 4 个区间：

(1) 测试用例 1：输入 4，返回 2　　　　//区间 ii 和 A

(2) 测试用例 2：输入−10，返回 0，输出"平方根错误，输入值小于 0"
　　　　//区间 i 和 B

上例的对等区间划分是非常简单的。当软件变得更加复杂时，对等区间的确定就越难，区间之间的相互依赖性就越强，使用对等区间划分设计测试用例技术的难度会增加。

3.2.3　等价类划分法测试用例

在某网站申请免费信箱时，要求用户必须输入用户名、密码及确认密码，对每一项输入条件的要求如下：

用户名要求为 4～16 位，使用英文字母、数字、“-”、“_”，并且首字符必须为字母或数字；密码要求为 6～16 位，只能使用英文字母、数字以及“-”、“_”，并且区分大小写。

分析如下：

（1）分析程序的规格说明，列出等价类表（包括有效等价类和无效等价类），如表 3-5 所示。

表 3-5　等价类表

输入条件	有效等价类	编号	输入条件	无效等价类	编号
用户名	4～16 位	1	用户名	少于 4 位	8
				多于 16 位	9
	首字符为字母	2		首字符为除字母、数字之外的其他字符	10
	首字符为数字	3			
	英文字母、数字、“-”、“_”组合	4		组合中含有除英文字母、数字、“-”、“_”之外的其他特殊字符	11
密码	6～16 位	5	密码	少于 6 位	12
				多于 16 位	13
	英文字母、数字、“-”、“_”组合	6		组合中含有除英文字母、数字、“-”、“_”之外的其他特殊字符	14
确认密码	内容与密码相同	7	确认密码	内容与密码相同，但字母大小写不同	15

（2）根据上述等价类表，设计测试用例如表 3-6 所示。

表 3-6　测试用例

测试用例	用户名	密码	确认密码	预期输出
TC1	abc_2000	abc_123	abc_123	注册成功
TC2	2000-abc	123-abc	123-abc	注册成功
TC3	abc	12345678	12345678	提示用户名错误
TC4	abcdefghijk123456	12345678	12345678	提示用户名错误
TC5	_abc123	12345678	12345678	提示用户名错误
TC6	abc&123	12345678	12345678	提示用户名错误
TC7	abc_123	12345	12345	提示密码错误
TC8	abc_123	abcdefghijk123456	abcdefghijk123456	提示密码错误
TC9	abc_123	abc&123	abc&123	提示密码错误
TC10	abc_123	abc_123	abc_123	提示密码错误

3.3 边界值分析法

3.3.1 边界值分析法的概念

边界值分析法(Boundary Value Analysis,BVA)是一种补充等价类划分法的测试用例设计技术,它不是选择等价类的任意元素,而是选择等价类边界的测试用例。在测试过程中,可能会忽略边界值的条件,而软件设计中大量的错误是发生在输入或输出范围的边界上,而不是发生在输入输出范围的内部。因此针对各种边界情况设计测试用例,可以查出更多的错误。

在实际的软件设计过程中,会涉及大量的边界值条件和过程,这里有一个简单的 VB 程序的例子:

```
Dim num(10) as Integer
Dim i as Integer
For i=1 to 10
   num(i)=1
Next i
```

在这个程序中,目标是为了创建一个拥有 10 个元素的一维数组,看似合理,但是,在大多数 Basic 语言中,当一个数组被定义时,其第一个元素所对应的数组下标是 0 而不是 1。由此,上述程序运行结束后,数组中成员的赋值情况如下:

num(0)=0,num(1)=1,num(2)=1,…,num(10)=1

这时,如果其他程序员在使用这个数组的时候,可能会造成软件的缺陷或者错误的产生。

使用边界值分析方法设计测试用例,首先应确定边界情况。通常输入和输出等价类的边界,就是应着重测试的边界情况。应当选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据,而不是选取等价类中的典型值或任意值作为测试数据。

在应用边界值分析法设计测试用例时,应遵循以下几条原则:

(1) 如果输入条件规定了值的范围,则应该选取刚达到这个范围的边界值,以及刚刚超过这个范围边界的值作为测试输入数据。

(2) 如果输入条件规定了值的个数,则用最大个数、最小个数、比最小个数少 1、比最大个数多 1 的数作为测试数据。

(3) 根据规格说明的每一个输出条件,分别使用以上两个原则。

(4) 如果程序的规格说明给出的输入域或者输出域是有序集合(如有序表、顺序文件等),则应选取集合的第一个元素和最后一个元素作为测试用例。

（5）如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界值作为测试用例。

（6）分析规格说明，找出其他可能的边界条件。

举例说明如下：

考虑学生考试成绩的输入（不计小数点），其输入数据是一个有限范围的整数，可以确定输入数据的最小值（min）和最大值（max），则有效的数据范围是 min≤N≤max，即 0≤N≤100。于是，可以选取输入变量的最小值（min）、略大于最小值（min＋1）、略小于最大值（max－1）和最大值（max）来设计测试用例。因此，学生分数的边界值分析法的有效测试数据是：0，1，99，100。有时，为了检查输入数据超过极限值时系统的情况，还需要考虑采用一个略超过最大值（max＋1）以及略小于最小值（min－1）的取值，即健壮性测试。所以，上述学生分数输入的无效测试数据为：－1，101。

3.3.2　边界条件与次边界条件

边界值分析法是对输入的边界值进行测试。在测试用例设计中，需要对输入的条件进行分析并且找出其中的边界值条件，通过对这些边界值的测试来查出更多的错误。

提出边界条件时，一定要测试临近边界的有效数据，测试最后一个可能有效的数据，同时测试刚超过边界的无效数据。通常情况下，软件测试所包含的边界检验有几种类型：数值、字符、位置、数量、速度、尺寸等。在设计测试用例时要考虑边界检验的类型的特征：第一个/最后一个、开始/完成、空/满、最大值/最小值、最快/最慢、最高/最低、最长/最短等。这些不是确定的列表，而是一些可能出现的边界条件。

举个例子来说明一下，如表 3-7 所示。

表 3-7　利用边界值作为测试数据的例子

项	边 界 值	测试用例的设计思路
字符	起始－1 个字符/结束＋1 个字符	假设一个文本输入区域要求允许输入 1 到 255 个字符，输入 1 个和 255 个字符作为有效等价类；输入 0 个和 256 个字符作为无效等价类，这几个数值都属于边界条件值
数值	开始位－1/结束位＋1	假设软件要求输入的数据为 5 位数值，则可以使用 00000 作为最小值和 99999 作为最大值，然后使用刚好小于 5 位和大于 5 位的数值来作为边界条件
方向	刚刚超过/刚刚低于	
空间	小于空余空间一点/大于满空间一点	假如要做磁盘的数据存储，使用比最小剩余磁盘空间大一点（几 KB）的文件作为最大值的检验边界条件

在多数情况下，边界值条件是基于应用程序的功能设计而需要考虑的因素，可以从软件的规格说明或常识中得到，也是最终用户通常最容易发现的问题。然而，在测试用例设计过程中，某些边界值条件是不需要呈现给用户的，或者说用户很难注意到这些问题，但这些边界条件同时确实属于检验范畴内的边界条件，称为内部边界值条件或次边界值条件。主要有下面三种。

1. 数值的边界值检验

计算机是基于二进制进行工作的，因此，任何数值运算都有一定的范围限制，如表 3-8 所示。

表 3-8　计算机数值运算的范围

项	范围或值
位(bit)	0 或 1
字节(byte)	0～255
字(word)	0～65、535(单字)或 0～4、294、967、295(双字)
千(K)	1024
兆(M)	1048576
吉(G)	1073741824
太(T)	1099511627776

例如对字节进行检验，边界值条件可以设置成 254、255 和 256。

2. 字符的边界值检验

在字符的编码方式中，ASCII 和 Unicode 是比较常见的编码方式，表 3-9 中列出了一些简单的 ASCII 码对应表。

表 3-9　字符的 ASCII 码对应表

字　符	ASCII 码值	字　符	ASCII 码值
空(null)	0	A	65
空格(space)	32	a	97
斜杠(/)	47	左中括号([)	91
0	48	z	122
冒号(：)	58	Z	90
@	64	单引号(')	96

在进行文本输入或者文本转换的测试过程中，需要非常清晰地了解 ASCII 码的一些基本对应关系，例如小写字母 z 和大写字母 Z 在表中的对应是不同的，这些也必须被考虑到数据区域划分的过程中，从而定义等价有效类，来设计测试用例。

3. 其他边界值检验

包括默认值/空值/空格/未输入值/零、无效数据/不正确数据和干扰数据等。

在实际的测试用例设计中，需要将基本的软件设计要求和程序定义的要求结合起来，即结合基本边界值条件和子边界值条件来设计有效的测试用例。

3.3.3 边界值分析法测试用例

某程序要求输入三个整数 x，y，z，分别作为长方体的长、宽、高，x，y，z 的取值范围均为 2～20，试计算长方体的体积。表 3-10 给出了健壮性边界值分析测试用例。

表 3-10 健壮性边界值分析测试用例

测试用例	x	y	z	预期输出
TC1	1	10	10	x 值超出范围
TC2	2	10	10	200
TC3	3	10	10	300
TC4	10	10	10	1000
TC5	19	10	10	1900
TC6	20	10	10	2000
TC7	21	10	10	x 值超出范围
TC8	10	1	10	y 值超出范围
TC9	10	2	10	200
TC10	10	3	10	300
TC11	10	19	10	1900
TC12	10	20	10	2000
TC13	10	21	10	y 值超出范围
TC14	10	10	1	z 值超出范围
TC15	10	10	2	200
TC16	10	10	3	300
TC17	10	10	19	1900
TC18	10	10	20	2000
TC19	10	10	21	z 值超出范围

3.4 决策表法

3.4.1 决策表法的概念

在所有的黑盒测试方法中，基于决策表（也称判定表）的测试是最为严格、最具有逻辑性的测试方法。决策表是分析和表达多个逻辑条件下执行不同操作情况的工具。由于决

策表可以把复杂的逻辑关系和多种条件组合的情况表达得既具体又明确，因此在程序设计发展的初期，决策表就已被当作编写程序的辅助工具了。

决策表通常由四个部分组成，如图 3-1 所示。

(1) 条件桩：列出了问题的所有条件，通常认为列出的条件的先后次序无关紧要。

(2) 动作桩：列出了问题规定的可能采取的操作，这些操作的排列顺序没有约束。

(3) 条件项：针对条件桩给出的条件列出所有可能的取值。

(4) 动作项：与条件项紧密相关，列出在条件项的各组取值情况下应该采取的动作。

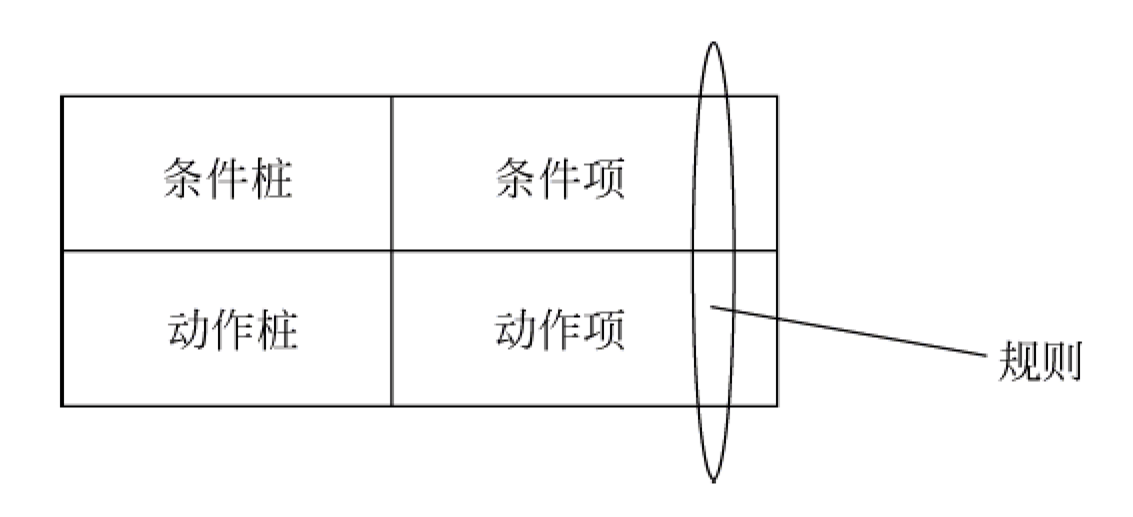


图 3-1　决策表的组成

任何一个条件组合的特定取值及其相应要执行的操作称为一条规则，在决策表中贯穿条件项和动作项的一列就是一条规则。显然，决策表中列出多少组条件取值，也就有多少条规则，即条件项和动作项有多少列。

根据软件规格说明，建立决策表的步骤如下：

(1) 确定规则的个数。假如有 n 个条件，每个条件有两个取值，故有 2^n 种规则。

(2) 列出所有的条件桩和动作桩。

(3) 填入条件项。

(4) 填入动作项，得到初始决策表。

(5) 化简。合并相似规则(相同动作)。

以下列问题为例给出构造决策表的具体过程。

如果某产品销售好并且库存低，则增加该产品的生产；如果该产品销售好，但库存量不低，则继续生产；若该产品销售不好，但库存量低，则继续生产；若该产品销售不好，且库存量不低，则停止生产。

解法如下：

(1) 确定规则的个数。对于本题有 2 个条件(销售、库存)，每个条件可以有两个取值，故有 $2^2=4$ 种规则。

(2) 列出所有的条件桩和动作桩。

(3) 填入条件项。

(4) 填入动作项，得到初始决策表，如表 3-11 所示。

表 3-11　决策表

选项＼规则		1	2	3	4
条件	C1：销售好？	T	T	F	F
	C2：库存低？	T	F	T	F
动作	a1：增加生产	√			
	a2：继续生产		√	√	
	a3：停止生产				√

每种测试方法都有适用的范围，决策表法适用于下列情况：

(1) 规格说明以决策表形式给出，或很容易转换成决策表。

(2) 条件的排列顺序不会也不应影响执行哪些操作。

(3) 规则的排列顺序不会也不应影响执行哪些操作。

(4) 每当某一规则的条件已经满足，并确定要执行的操作后，不必检验别的规则。

(5) 如果某一规则得到满足要执行多个操作，这些操作的执行顺序无关紧要。

3.4.2　决策表法的应用

决策表最突出的优点是，能够将复杂的问题按照各种可能的情况全部列举出来，简明并避免遗漏。因此，利用决策表能够设计出完整的测试用例集合。运用决策表设计测试用例，可以将条件理解为输入，将动作理解为输出。

例如，某股票公司的佣金政策如下：如果一次销售额少于 1000 元，那么基础佣金将是销售额的 8.4%；如果销售额大于 1000 元，但少于 10000 元，那么基础佣金将是销售额的 5%，外加 34 元；如果销售额大于 10000 元，那么基础佣金将是销售额的 4%，外加 134 元。另外销售单价和销售的份数对佣金也有影响。如果单价低于 14 元/份，则外加基础佣金的 5%，此外若不是整百的份数，再外加 4%的基础佣金；若单价在 14 元/份以上，但低于 25 元/份，则加 2%的基础佣金，若不是整百的份数，再外加 4%的基础佣金；若单价在 25 元/份以上，并且不是整百的份数，则外加 4%的基础佣金。

分析如下：

(1) 分析各种输入情况，列出为输入变量销售额、单价、份数划分的有效等价类。

销售额的有效等价类：

S1：$\{0 \leqslant Sale < 1000\}$

S2：$\{1000 \leqslant Sale < 10000\}$

S3：$\{Sale \geqslant 10000\}$

单价的有效等价类：

P1：$\{Price < 14\}$

P2：$\{14 \leqslant Price < 25\}$

P3：{Price≥25}

份数的有效等价类：

B1：{整百的份数}

B2：{不是整百的份数}

(2) 分析程序的规格说明，并结合以上等价类划分的情况，给出问题规定的可能采取的操作(即列出所有的动作桩)。考虑各种有效的输入情况，程序中可能采取的操作有以下 6 种：

a1：基础佣金为销售额的 8.4%

a2：基础佣金为销售额的 5%+34 元

a3：基础佣金为销售额的 4%+134 元

a4：基础佣金的 5%

a5：基础佣金的 2%

a6：基础佣金的 4%

(3) 根据以上分析的步骤(1)和(2)，画出决策表，如表 3-12 所示。

表 3-12 股票佣金政策的决策表

选项 \ 规则		1	2	3	4	5	6	7	8	9
条件	C1：销售额	S1	S1	S1	S1	S1	S1	S2	S2	S2
	C2：单价	P1	P1	P2	P2	P3	P3	P1	P1	P2
	C3：份数	B1	B2	B1	B2	B1	B2	B1	B2	B1
动作	a1	√	√	√	√	√	√			
	a2							√	√	√
	a3									
	a4	√	√					√	√	
	a5			√	√					√
	a6		√		√		√		√	

选项 \ 规则		10	11	12	13	14	15	16	17	18
条件	C1：销售额	S2	S2	S2	S3	S3	S3	S3	S3	S3
	C2：单价	P2	P3	P3	P1	P1	P2	P2	P3	P3
	C3：份数	B2	B1	B2	B1	B2	B1	B2	B1	B2
动作	a1									
	a2	√	√	√						
	a3				√	√	√	√	√	√
	a4				√	√				
	a5	√					√	√		
	a6	√		√		√		√		√

根据决策表 3-12,可设计测试用例,如表 3-13 所示。

表 3-13　股票佣金政策的测试用例表

测试用例	销售额(元)	单价(元/份)	份数(份)	预期输出(元)
TC1	800	12	200	70.56
TC2	800	12	240	73.25
TC3	800	20	200	68.54
TC4	800	20	240	71.23
TC5	800	30	200	67.2
TC6	800	30	240	69.89
TC7	6000	12	500	350.7
TC8	6000	12	750	364.06
TC9	6000	20	500	340.68
TC10	6000	20	750	354.04
TC11	6000	30	500	334
TC12	6000	30	750	347.36
TC13	20000	12	4000	980.7
TC14	20000	12	4590	1018.06
TC15	20000	20	4000	952.68
TC15	20000	20	4590	990.04
TC17	20000	30	4000	934
TC18	20000	30	4590	971.36

3.5　因果图法

3.5.1　因果图法的概念

前面介绍的等价类划分法和边界值分析法都着重考虑输入条件,而没有考虑到输入条件的各种组合情况,也没有考虑到各个输入条件之间的相互制约关系。因此,必须考虑采用一种适合于多种条件的组合,相应能产生多个动作的形式来进行测试用例的设计,这就需要采用因果图法。因果图法就是一种利用图解法分析输入的各种组合情况,从而设计测试用例的方法,它适合于检查程序输入条件的各种情况的组合。

在因果图中使用 4 种符号分别表示 4 种因果关系,如图 3-2 所示。用直线连接左右结点,其中左结点 C_i 表示输入状态(或称原因),右结点 e_i 表示输出状态(或称结果)。C_i 和 e_i 都可取值 0 或 1,0 表示某状态不出现,1 表示某状态出现。

图 3-2 中各符号的含义如下:

(a)图表恒等:若 C_1 是 1,则 e_1 也是 1;否则若 C_1 是 0,则 e_1 为 0。

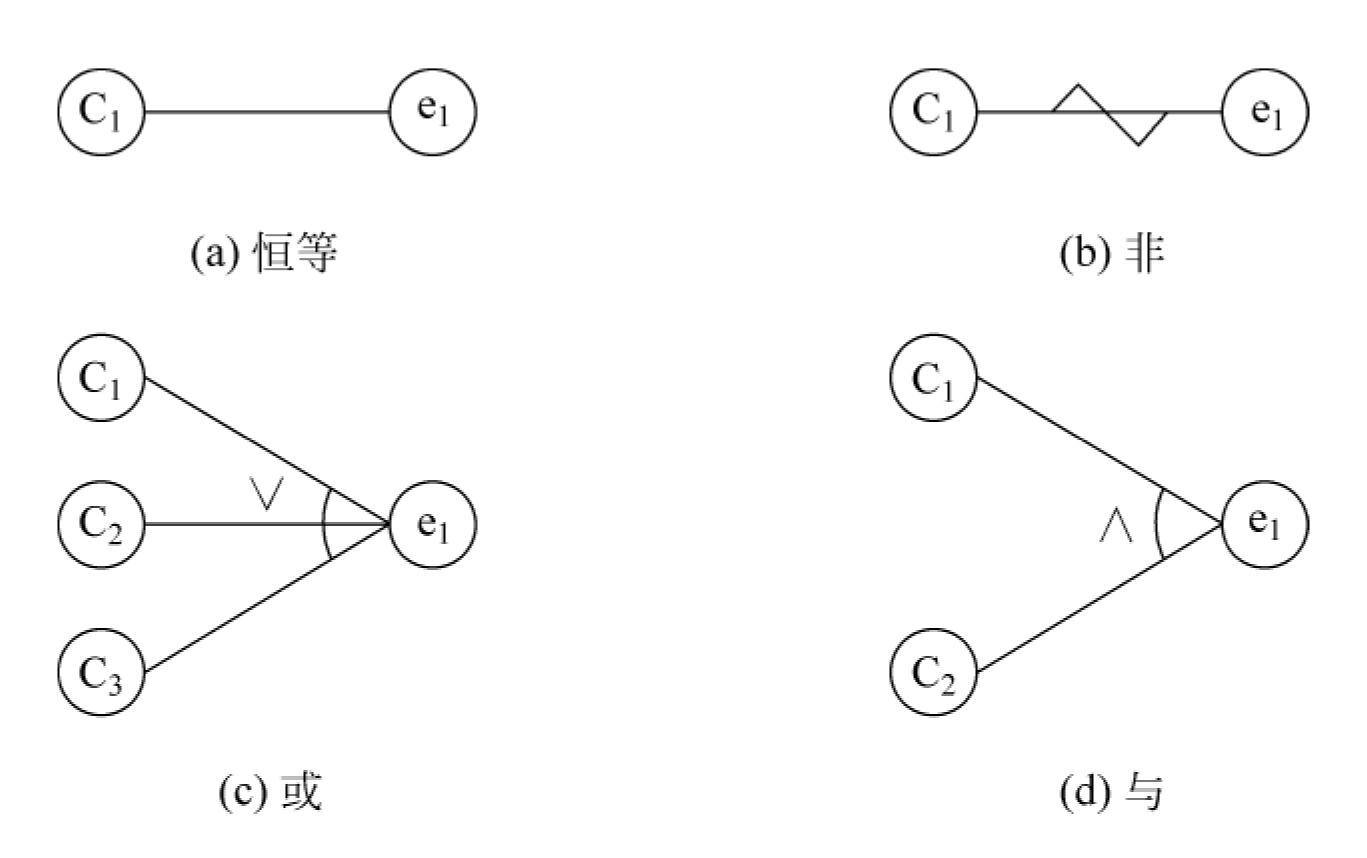


图 3-2 因果图的基本符号

(b)图表非：若 C_1 是 1，则 e_1 是 0；否则 C_1 是 0，则 e_1 为 1。

(c)图表或：若 C_1 或 C_2 或 C_3 是 1，则 e_1 是 1；否则若 C_1、C_2、C_3 全为 0，则 e_1 为 0。

(d)图表与：若 C_1 和 C_2 都是 1，则 e_1 是 1；否则只要 C_1、C_2、C_3 中有一个为 0，e_1 为 0。

在实际问题中，输入状态相互之间还可能存在某些依赖关系，我们称之为约束。例如，某些输入条件不可能同时出现。输出状态之间也往往存在约束，在因果图中，以特定的符号标明这些约束，如图 3-3 所示。

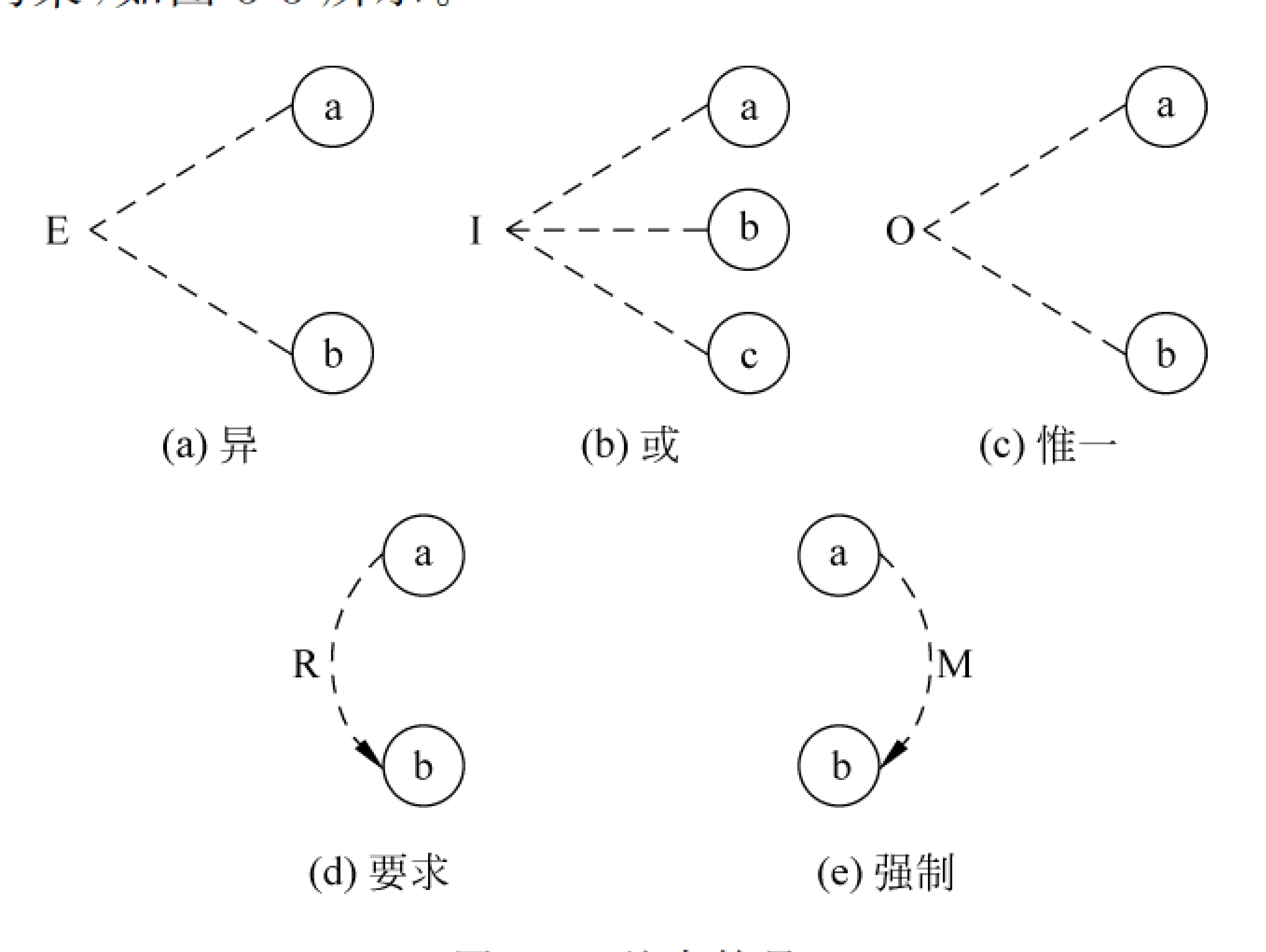


图 3-3 约束符号

对输入条件的约束有：

(a)图表 E 约束(异)：a 和 b 中最多有一个可能为 1，即 a 和 b 不能同时为 1。

(b)图表 I 约束(或)：a、b 和 c 中至少有一个必须是 1，即 a、b 和 c 不能同时为 0。

(c)图表 O 约束(惟一)：a 和 b 中必须有一个且仅有一个为 1。

(d)图表 R 约束(要求)：a 是 1 时，b 必须是 1，即 a 是 1 时，b 不能是 0。

对输出条件的约束只有 M 约束。

(e)图表 M 约束(强制)：若结果 a 是 1，则结果 b 强制为 0。

因果图法最终要生成决策表。

利用因果图法生成测试用例需要以下几个步骤：

(1) 分析软件规格说明书中的输入输出条件，并且分析出等价类。分析规格说明中的语义的内容，通过这些语义来找出相对应的输入与输入之间，输入与输出之间的对应关系。

(2) 将对应的输入与输入之间，输入与输出之间的关系连接起来，并且将其中不可能的组合情况标注成约束或者限制条件，形成因果图。

(3) 将因果图转换成决策表。

(4) 将决策表的每一列作为依据，设计测试用例。

各步骤如图 3-4 所示。

从因果图生成的测试用例中包括了所有输入数据取 True 与 False 的情况，构成的测试用例数目达到最少，且测试用例数目随输入数据数目的增加而线性地增加。

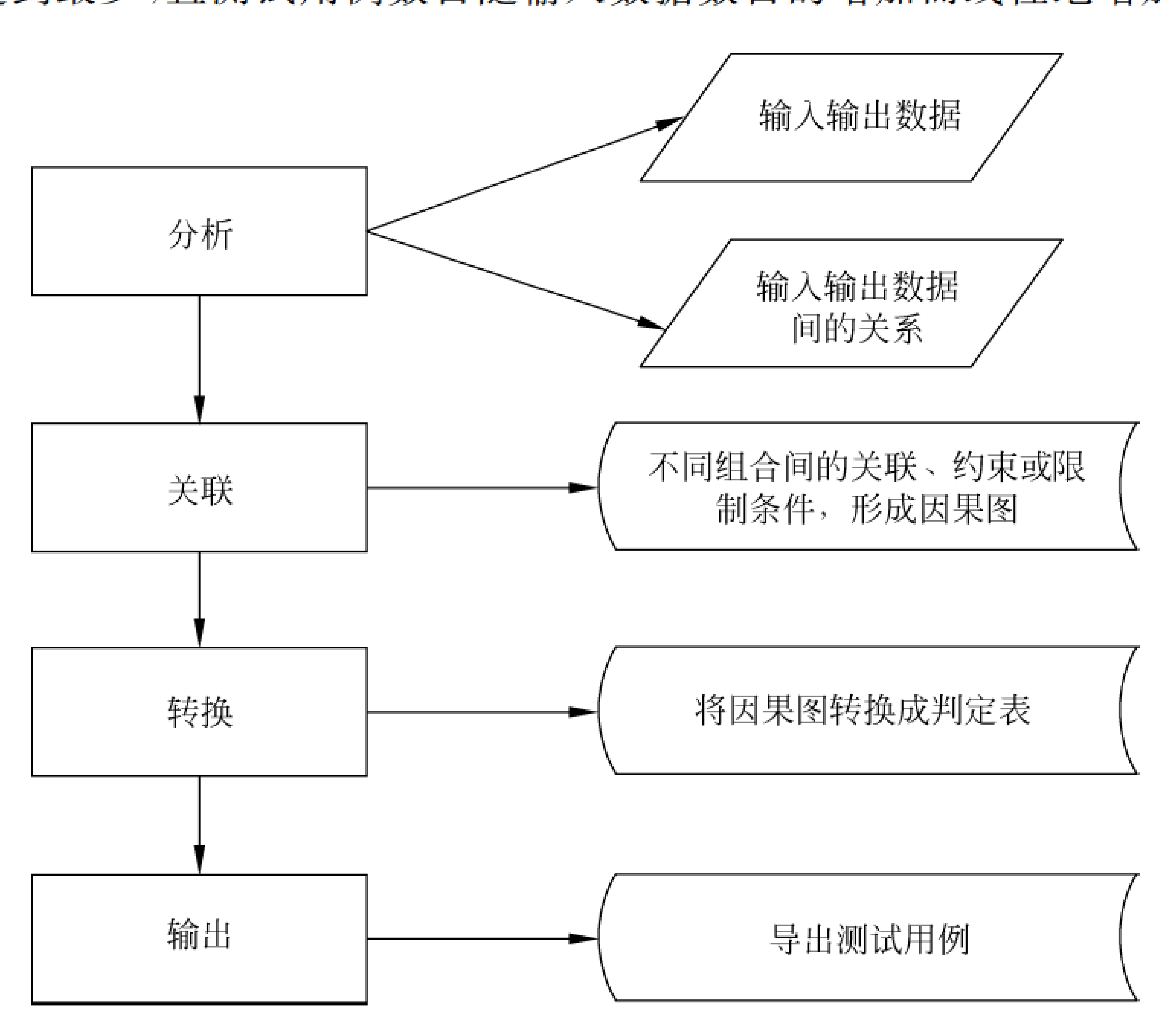


图 3-4　因果图法示例

3.5.2 因果图法测试用例

某软件规格说明中包含这样的要求：输入的第一个字符必须是 A 或 B，第二个字符必须是一个数字，在此情况下进行文件的修改；但如果第一个字符不正确，则给出信息 L；如果第二个字符不是数字，则给出信息 M。

解法如下：

(1) 分析程序的规格说明，列出原因和结果。

原因：C_1——第一个字符是 A

C_2——第一个字符是 B

C_3——第二个字符是一个数字

结果：e_1——给出信息 L

e_2——修改文件

e_3——给出信息 M

(2) 将原因和结果之间的因果关系用逻辑符号连接起来，得到因果图，如图 3-5 所示。编号为 11 的中间结点是导出结果的进一步原因。

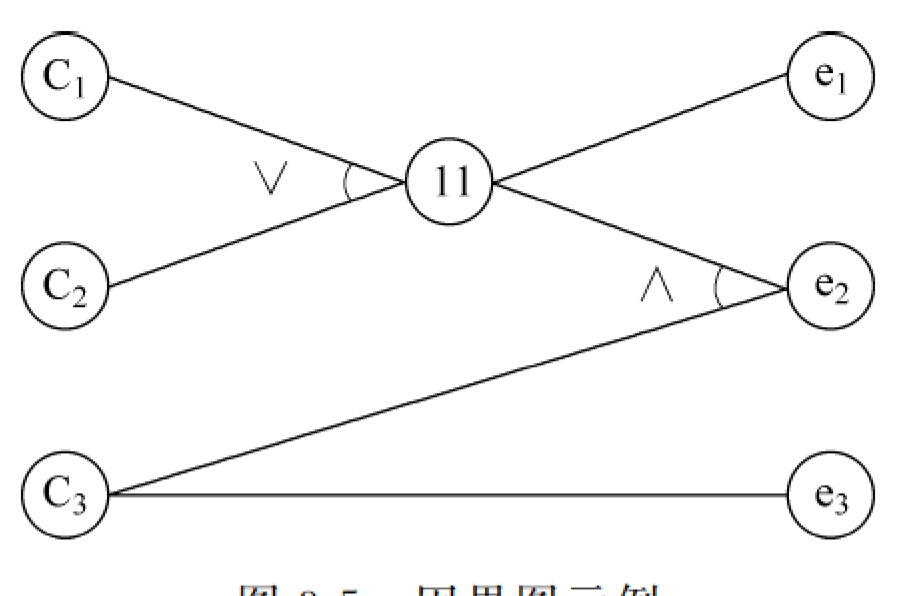


图 3-5 因果图示例

因为 C_1 和 C_2 不可能同时为 1，即第一个字符不可能既是 A 又是 B，在因果图上可对其施加 E 约束，得到具有约束的因果图，如图 3-6 所示。

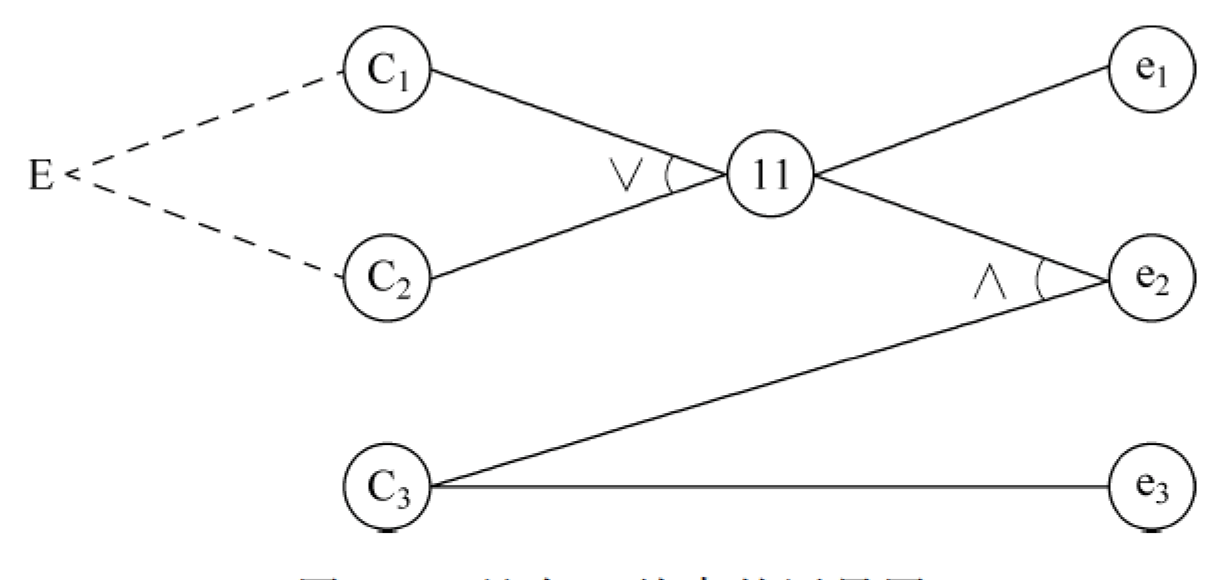


图 3-6 具有 E 约束的因果图

(3) 将因果图转换成决策表，如表 3-14 所示。

(4) 设计测试用例。表 3-14 中的前两种情况，因为 C_1 和 C_2 不可能同时为 1，所以应排除这两种情况。根据此表，可以设计出 6 个测试用例，如表 3-15 所示。

表 3-14　决策表

选项 \ 规则		1	2	3	4	5	6	7	8
条件	C1	1	1	1	1	0	0	0	0
	C2	1	1	0	0	1	1	0	0
	C3	1	0	1	0	1	0	1	0
	11			1	1	1	1	0	0
动作	e1			0	0	0	0	1	1
	e2			1	0	1	0	0	0
	e3			0	1	0	1	0	1
	不可能	1	1						
测试用例				A5	A#	B9	B?	X2	Y%

表 3-15　测试用例

编号	输入数据	预 期 输 出
TC1	A5	修改文件
TC2	A#	给出信息 M
TC3	B9	修改文件
TC4	B?	给出信息 M
TC5	X2	给出信息 L
TC6	Y%	给出信息 L 和信息 M

事实上，在较为复杂的问题中，因果图法非常有效，可以帮助检查输入条件组合，设计出非冗余、高效的测试用例。如果开发项目在设计阶段就采用了决策表，就不必再画因果图，可以直接利用决策表设计测试用例。

3.6　实例设计

本节将以成绩管理系统为例，介绍在实际项目中如何来做黑盒测试。测试用例的设计方法不是单独存在的，具体到每个测试项目里都会用到多种方法，在实际测试中，往往是综合使用各种方法才能高效率、高质量的完成测试。一个好的测试策略和测试方法必将给整个测试工作带来事半功倍的效果，从而充分利用有限的人力和物力资源。

在本系统中，登录窗口的界面如图 3-7 所示，成绩录入窗口的界面如图 3-8 所示。



图 3-7　系统登录界面

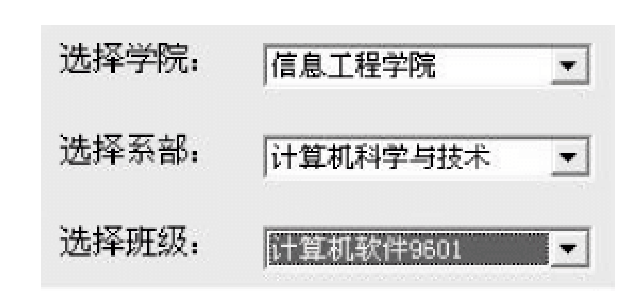


图 3-8　成绩录入界面

在登录窗口中不考虑身份选择情况，只验证用户名和密码以及登录、重填按钮的正确性。用户名和密码的输入条件均要求为不超过 16 位，可以使用汉字、英文字母和数字及各种组合。

首先应用等价类划分法对用户名和密码进行等价类划分（包括有效等价类和无效等价类），如表 3-16 所示。

表 3-16　登录窗口的等价类表

输入条件	有效等价类	编号	输入条件	无效等价类	编号
用户名	1～16 位	1	用户名	空值	5
				多于 16 位	6
	英文字母、数字、汉字组合	2		组合中含有除英文字母、数字、汉字之外的其他字符	7
密码	1～16 位	3	密码	空值	8
				多于 16 位	9
	英文字母、数字、汉字组合	4		组合中含有除英文字母、数字、汉字之外的其他字符	10

登录窗口除了要验证用户名和密码的有效性，还要验证各个功能之间的正确性，因此，再应用决策表法。登录窗口对应的决策表如表 3-17 所示。

表 3-17　登录窗口的决策表

选项	规则	1	2	3	4	5	6	7
条件	C1：用户名正确？	—	—	—	1	1	0	0
	C2：密码正确？	—	—	—	1	0	1	0
	C3：选择登录按钮	1	0	0	1	1	1	1
	C4：选择重填按钮	1	0	1	0	0	0	0
动作	a1：提示错误					√	√	√
	a2：成功登录				√			
	a3：重填选项	√		√				
	不可能		√					

根据上述分析，可以确定测试用例如表 3-18 所示。

表 3-18　登录窗口测试用例

<table>
<tr><td>项目/软件名称</td><td>成绩管理系统</td><td>程序版本</td><td colspan="2">1.0</td></tr>
<tr><td>功能模块名</td><td>LOGIN</td><td>编制人</td><td colspan="2">×××</td></tr>
<tr><td>用例编号</td><td>SYSTEM_LOGIN_1</td><td>编制时间</td><td colspan="2">2006.3.25</td></tr>
<tr><td>相关的用例</td><td>无</td><td>预置条件</td><td colspan="2">无</td></tr>
<tr><td>功能特性</td><td colspan="4">用户身份验证</td></tr>
<tr><td>测试目的</td><td colspan="4">验证是否输入合法的信息，允许合法登录，阻止非法登录</td></tr>
<tr><td>预置条件</td><td>无</td><td>特殊规程说明</td><td colspan="2">如数据库访问权限</td></tr>
<tr><td>参考信息</td><td colspan="4">需求说明中关于“登录”的说明</td></tr>
<tr><td>测试数据</td><td colspan="4">用户名：abcd　密码：123</td></tr>
<tr><td>操作步骤</td><td>操作描述</td><td>数　据</td><td>期望结果</td><td>实际结果</td></tr>
<tr><td>1</td><td>输入用户名和密码，单击“登录”按钮</td><td>用户名：abcd
密码：123</td><td>成功登录，进入系统页面</td><td>一致</td></tr>
<tr><td>2</td><td>输入密码，单击“登录”按钮</td><td>用户名：空
密码：123</td><td>提示“用户名错误！”</td><td>一致</td></tr>
<tr><td>3</td><td>输入用户名和密码，单击“登录”按钮</td><td>用户名：abcdefghijk123456
密码：123</td><td>提示“用户名错误！”</td><td>一致</td></tr>
<tr><td>4</td><td>输入用户名和密码，单击“登录”按钮</td><td>用户名：abcd#
密码：123</td><td>提示“用户名错误！”</td><td>一致</td></tr>
<tr><td>5</td><td>输入用户名和密码，单击“登录”按钮</td><td>用户名：abcd
密码：空</td><td>提示“密码错误！”</td><td>一致</td></tr>
<tr><td>6</td><td>输入用户名和密码，单击“登录”按钮</td><td>用户名：abcd
密码：abcdefghijk123456</td><td>提示“密码错误！”</td><td>一致</td></tr>
<tr><td>7</td><td>输入用户名和密码，单击“登录”按钮</td><td>用户名：abcd
密码：123*</td><td>提示“密码错误！”</td><td>一致</td></tr>
<tr><td>8</td><td>输入用户名和密码，单击“登录”按钮</td><td>用户名：abcd
密码：321</td><td>提示“密码错误！”</td><td>一致</td></tr>
<tr><td>9</td><td>输入用户名和密码，单击“登录”按钮</td><td>用户名：dcba
密码：123</td><td>提示“用户名错误！”</td><td>一致</td></tr>
<tr><td>10</td><td>输入用户名和密码，单击“登录”按钮</td><td>用户名：dcba
密码：321</td><td>提示“用户名错误！”</td><td>一致</td></tr>
<tr><td>11</td><td>输入用户名和密码，单击“登录”按钮</td><td>用户名：空
密码：空</td><td>提示“用户名错误！”</td><td>一致</td></tr>
<tr><td>12</td><td>输入用户名和密码，单击“重填”按钮</td><td>用户名：abcd
密码：123</td><td>清空所有输入信息</td><td>一致</td></tr>
</table>

成绩录入窗口的需求规格说明如下：界面应包括三个下拉列表，分别用于显示各学院名称、各系部名称及各班级名称。只有选择了某一个学院后，系部列表框才为可用，列表中将显示出所选择学院对应的所有系部；同样，只有选择了某一个学院后，又选择了某一个系部，此时班级列表框才为可用，列表中将显示出所选择系部对应的所有班级。当三个选项都已经完成选择后，界面上则会显示出所选班级的名单，这时就可以录入成绩了。

对于本例，可以利用决策表的方法来设计测试用例，步骤如下：

(1) 建立决策表。由规格说明可以分析出，基本输入事件有：

① C1：选择学院

② C2：选择系部

③ C3：选择班级

基本输出事件有：

① a1：显示所选班级名单

② a2：学院列表框可用

③ a3：系部列表框可用

④ a4：班级列表框可用

⑤ a5：显示各学院名称

⑥ a6：显示各系部名称

⑦ a7：显示各班级名称

⑧ a8：不能显示具体选项(如在没有选择学院的前提下，系部列表框中将不能显示所对应的具体系部)

根据上述分析得出的输入事件、输出事件建立决策表，如表 3-19 所示。

表 3-19　输入和输出决策表

选项 \ 规则		1	2	3	4	5	6	7	8
条件	C1：选择学院	T	T	T	T	F	F	F	F
	C2：选择系部	T	T	F	F	T	T	F	F
	C3：选择班级	T	F	T	F	T	F	T	F
动作	a1：显示所选班级名单	√							
	a2：学院列表框可用		√	√	√	√	√	√	√
	a3：系部列表框可用		√	√	√				
	a4：班级列表框可用		√						
	a5：显示各学院名称		√	√	√				
	a6：显示各系部名称								
	a7：显示各班级名称			√		√	√	√	
	a8：不能显示具体选项								

（2）确定测试用例，如表 3-20 所示。

表 3-20　成绩录入窗口测试用例

项目/软件名称	成绩管理系统	程序版本	1.0
功能模块名	INPUT	编制人	×××
用例编号	SYSTEM_INPUT_1	编制时间	2006.3.25
相关的用例	无	预置条件	无
功能特性	成绩录入		
测试目的	验证成绩录入窗口是否符合用户的需求		
预置条件	无	特殊规程说明	如数据库访问权限
参考信息	需求说明中关于“成绩录入”的说明		
测试数据	学院：信息工程学院；系部：计算机科学与技术；班级：计算机软件 9601		

操作步骤	操作描述	数　据	期 望 结 果	实际结果
1	依次单击并选择学院、系部及班级	学院：信息工程学院 系部：计算机科学与技术 班级：计算机软件 9601	三个列表框均可用，显示所选班级名单	一致
2	依次单击并选择学院、系部，单击班级列表框但并不选择其中列表项	学院：信息工程学院 系部：计算机科学与技术 班级：空	三个列表框均可用，学院列表框中显示信息工程学院，系部列表框中显示计算机科学与技术，班级列表中可以显示数据项	一致
3	依次单击并选择学院、班级，单击系部列表框但并不选择其中列表项	学院：信息工程学院 系部：空 班级：空	学院和系部列表可用，学院列表框中显示信息工程学院，系部列表框中可以显示数据项，班级列表中不能显示相应数据项	一致
4	单击并选择学院，单击系部和班级列表框但并不选择其中列表项	学院：信息工程学院 系部：空 班级：空	学院和系部列表可用，学院列表框中显示信息工程学院，系部列表框中可以显示数据项，班级列表中不能显示相应数据项	一致
5	单击但不选择学院，单击并选择系部和班级	学院：空 系部：空 班级：空	学院列表可用，系部和班级列表中不能显示相应数据项	一致
6	单击但不选择学院，单击并选择系部，单击但并不选择班级	学院：空 系部：空 班级：空	学院列表可用，系部和班级列表中不能显示相应数据项	一致

续表

操作步骤	操作描述	数　　据	期望结果	实际结果
7	单击但不选择学院，单击但并不选择系部，单击并选择班级	学院：空 系部：空 班级：空	学院列表可用，系部和班级列表中不能显示相应数据项	一致
8	单击但不选择学院	学院：空 系部：空 班级：空	学院列表可用	一致

3.7 测试方法的选择

为了最大限度地减少测试遗留的缺陷，同时也为了最大限度地发现存在的缺陷，在测试实施之前，测试工程师必须确定将要采用的测试策略和测试方法，并以此为依据制定详细的测试方案。通常，一个好的测试策略和测试方法必将给整个测试工作带来事半功倍的效果。

如何才能确定好的测试策略和测试方法呢？通常，在确定测试方法时，应该遵循以下原则：

(1) 根据程序的重要性和一旦发生故障将造成的损失程度来确定测试等级和测试重点。

(2) 认真选择测试策略，以便能尽可能少地使用测试用例，发现尽可能多的程序错误。因为一次完整的软件测试过后，如果程序中遗留的错误过多并且严重，则表明该次测试是不足的，而测试不足则意味着让用户承担隐藏错误带来的危险，但测试过度又会带来资源的浪费。因此，测试需要找到一个平衡点。

以下是各种测试方法选择的综合策略，可在实际应用过程中参考。

(1) 首先进行等价类划分，包括输入条件和输出条件的等价划分，将无限测试变成有限测试，这是减少工作量和提高测试效率的最有效方法。

(2) 在任何情况下都必须使用边界值分析方法。经验表明用这种方法设计出测试用例发现程序错误的能力最强。

(3) 对照程序逻辑，检查已设计出的测试用例的逻辑覆盖程度。如果没有达到要求的覆盖标准，应当再补充足够的测试用例。

(4) 如果程序的功能说明中含有输入条件的组合情况，则应在一开始就选用因果图法。

小　　结

测试用例的设计方法不是单独存在的，具体到每个测试项目里都会用到多种方法。每种类型的软件有各自的特点，每种测试用例设计的方法也有各自的特点，针对不同软件如何利用这些黑盒方法是非常重要的。在实际测试中，往往是综合使用各种方法才能有效提高测试效率和测试覆盖度，这就需要认真掌握这些方法的原理，积累更多的测试经验，以有效提高测试水平。

习　　题

1. 叙述黑盒测试技术的实质及要点。

2. 常用的黑盒测试用例设计方法有哪些？各有什么优缺点？

3. 使用等价类划分法设计三角形问题的测试用例。

4. 程序要求某个输入为 6 位正整数，试用不同的测试方法设计所有的测试用例。

5. 测试银行提款机上的提款功能，要求用户输入的提款金额的有效数值为 50～2000，并以 50 为最小单位（即取款金额为 50 的倍数），且小数点后为 00，除小数点外，不可以出现数字以外的任何符号和文字。试用等价类划分法和边界值分析法设计测试用例。

6. 某程序要求输入日期，规定变量 month、day、year 的取值范围为：1≤month≤12，1≤day≤31，1958≤year≤2058，试用边界值分析法设计测试用例。

第 4 章

白盒测试及其实例设计

本章概述

白盒测试是软件测试实践中最为有效和实用的方法之一。白盒测试是基于程序的测试，检测产品的内部结构是否合理以及内部操作是否按规定执行，覆盖测试与路径测试是其两大基本策略。本章重点围绕逻辑覆盖和路径分析这两种方法展开介绍常见的白盒测试方法，并通过实例说明如何实际运用白盒测试技术。

4.1 逻辑覆盖测试

白盒测试技术的常见方法之一就是覆盖测试，它是利用程序的逻辑结构设计相应的测试用例。测试人员要深入了解被测程序的逻辑结构特点，完全掌握源代码的流程，才能设计出恰当的用例。根据不同的测试要求，覆盖测试可以分为语句覆盖、判断覆盖、条件覆盖、判断/条件覆盖、条件组合覆盖和路径覆盖。

下面是一段简单的 C 语言程序，作为公共程序段来说明 6 种覆盖测试各自的特点。

程序 4-1：

```
If (x>100&& y>500) then
  score=score+1
If (x>=1000|| z>5000) then
  score=score+5
```

逻辑运算符“&&”表示“与”的关系，逻辑运算符“||”表示“或”的关系。其程序控制流程图如图 4-1 所示。

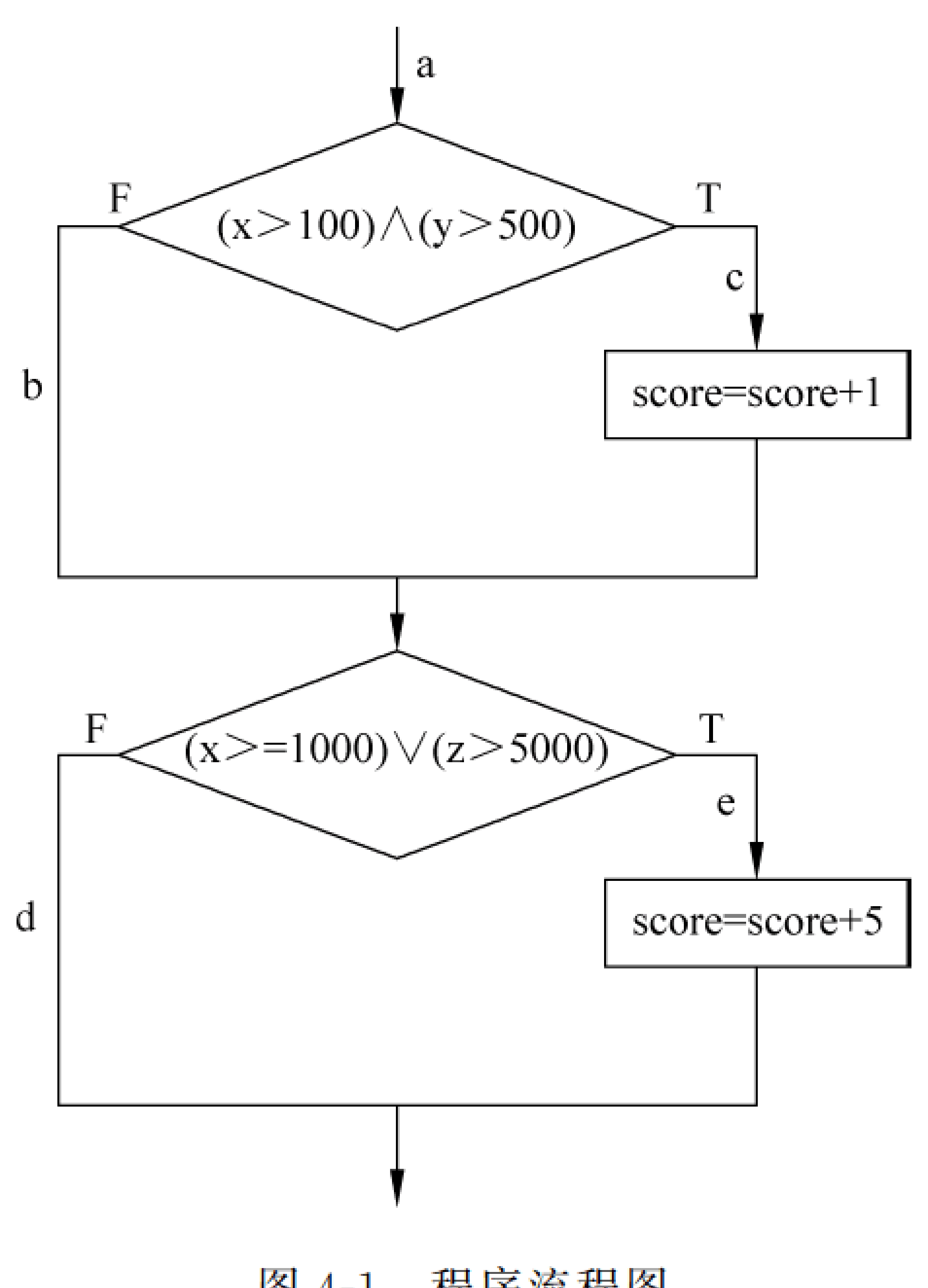


图 4-1　程序流程图

4.1.1　语句覆盖

语句覆盖(Statement Coverage)是指设计若干个测试用例,程序运行时每个可执行语句至少被执行一次。在保证完成要求的情况下,测试用例的数目越少越好。

以下是针对公共程序段设计的两个测试用例,称为测试用例组 1:

Test Case 1: x=2000,y=600,z=6000

Test Case 2: x=900,y=600,z=5000

如表 4-1 所示,采用 Test Case 1 作为测试用例,则程序按路径 a,c,e 顺序执行,程序中的 4 个语句都被执行一次,符合语句覆盖的要求。采用 Test Case 2 作为测试用例,则程序按路径 a,c,d 顺序执行,程序中的语句 4 没有执行到,所以没有达到语句覆盖的要求。

表 4-1　测试用例组 1

测试用例	x,y,z	(x>100)and (y>500)	(x>=1000)or (z>5000)	执行路径
Test Case 1	2000,600,6000	True	True	ace
Test Case 2	900,600,5000	True	False	acd

从表面上看,语句覆盖用例测试了程序中的每一个语句行,好像对程序覆盖得很全面,但实际上语句覆盖测试是最弱的逻辑覆盖方法。例如,如果第一个判断的逻辑运算符

“&&”错误写成“||”，或者第二个判断的逻辑运算符“||”错误写成“&&”，这时如果采用 Test Case 1 测试用例是检验不出程序中的判断逻辑错误的。如果语句 3“If (x>=1000|| z>5000) then”错误的写成“If (x>=1500|| z>5000) then”，Test Case 1 同样无法发现错误之处。

根据上述分析可知，语句覆盖测试只是表面上的覆盖程序流程，没有针对源程序各个语句间的内在关系，设计更为细致的测试用例。

4.1.2 判断覆盖

判断覆盖(Branch Coverage)是指设计若干个测试用例，执行被测试程序时，程序中每个判断条件的真值分支和假值分支至少被执行一遍。在保证完成要求的情况下，测试用例的数目越少越好。判断覆盖又称为分支覆盖。

测试用例组 2：

Test Case 1：x=2000，y=600，z=6000

Test Case 3：x=50，y=600，z=2000

如表 4-2 所示，采用 Test Case 1 作为测试用例，程序按路径 a，c，e 顺序执行；采用 Test Case 3 作为测试用例，程序按路径 a，b，d 顺序执行。所以采用这一组测试用例，公共程序段的 4 个判断分支 b，c，d，e 都被覆盖到了。

表 4-2 测试用例组 2

测试用例	x，y，z	(x>100)and (y>500)	(x>=1000)or (z>5000)	执行路径
Test Case 1	2000，600，6000	True	True	ace
Test Case 3	50，600，2000	False	False	abd

测试用例组 3：

Test Case 4：x=2000，y=600，z=2000

Test Case 5：x=2000，y=200，z=6000

如表 4-3 所示，采用 Test Case 作为测试用例，程序按路径 a，c，d 顺序执行；采用 Test Case 5 作为测试用例，程序按路径 a，b，e 顺序执行。显然采用这组测试用例同样可以满足判断覆盖。

表 4-3 测试用例组 3

测试用例	x，y，z	(x>100)and (y>500)	(x>=1000)or (z>5000)	执行路径
Test Case 4	2000，600，2000	True	False	acd
Test Case 5	2000，200，6000	False	True	abe

实际上，测试用例组 2 和测试用例组 3 不仅达到了判断覆盖要求，也同时满足了语句覆盖要求。某种程度上可以说判断覆盖测试要强于语句覆盖测试。但是，如果将第二个判断条件((x>=1000)or (z>5000))中的 z>5000 错误定义成 z 的其他限定范围，由于判断条件中的两个判断式是“或”的关系，其中一个判断式错误是不影响结果的，所以这两组测试用例是发现不了问题的。因此，应该用具有更强逻辑覆盖能力的覆盖测试方法来测试这种内部判断条件。

4.1.3　条件覆盖

条件覆盖(Condition Coverage)是指设计若干个测试用例，执行被测试程序时，程序中每个判断条件中的每个判断式的真值和假值至少被执行一遍。

测试用例组 4：

Test Case 1：x=2000，y=600，z=6000

Test Case 3：x=50，y=600，z=2000

Test Case 5：x=2000，y=200，z=6000

如表 4-4 所示，把前面设计过的测试用例挑选出 Test Case 1，Test Case 3，Test Case 5 组合成测试用例组 4，组中的 3 个测试用例覆盖了 4 个内部判断式的 8 种真假值情况。同时这组测试用例也实现了判断覆盖。但是并不可以说判断覆盖是条件覆盖的子集。

测试用例组 5：

Test Case 6：50，600，6000

Test Case 7：2000，200，1000

如表 4-5(a) 和表 4-5(b)所示，其中表 4-5(a)表示每个判断条件的每个判断式的真值和假值，表 4-5(b)表示每个判断条件的真值和假值。测试用例组 5 中的 2 个测试用例虽然覆盖了 4 个内部判断式的 8 种真假值情况，但是这组测试用例的执行路径是 abe，仅仅覆盖了判断条件的 4 个真假分支中的 2 个。所以，需要设计一种能同时满足判断覆盖和条件覆盖的覆盖测试方法，即判断/条件覆盖测试。

表 4-4　测试用例组 4

测试用例	x，y，z	(x>100)	(y>500)	(x>=1000)	(z>5000)	执行路径
Test Case 1	2000，600，6000	True	True	True	False	ace
Test Case 3	50，600，2000	False	True	False	False	abd
Test Case 5	2000，200，6000	True	False	True	True	abe

表 4-5(a)　测试用例组 5

测试用例	x，y，z	(x>100)	(y>500)	(x>=1000)	(z>5000)	执行路径
Test Case 6	50，600，6000	False	True	False	True	abe
Test Case 7	2000，200，1000	True	False	True	False	abe

表 4-5(b)　测试用例组 5

测试用例	x,y,z	(x>100)and (y>500)	(x>=1000)or (z>5000)	执行路径
Test Case 6	50,600,6000	False	True	abe
Test Case 7	2000,200,1000	False	True	abe

4.1.4　判断/条件覆盖

判断/条件覆盖是指设计若干个测试用例,执行被测试程序时,程序中每个判断条件的真假值分支至少被执行一遍,并且每个判断条件内部的判断式的真假值分支也要被执行一遍。

测试用例组 6:

Test Case 1: x=2000, y=600, z=2000

Test Case 6: x=2000, y=200, z=6000

Test Case 7: x=2000, y=600, z=2000

Test Case 8: x=50, y=200, z=2000

如表 4-6(a) 和表 4-6(b)所示,其中表 4-6(a)表示每个判断条件的每个判断式的真值和假值,表 4-6(b)表示每个判断条件的真值和假值。测试用例组 6 虽然满足了判断覆盖和条件覆盖,但是没有对每个判断条件的内部判断式的所有真假值组合进行测试。条件组合判断是必要的,因为条件判断语句中的"与"即 and 和"或"即 or 会使内部判断式之间产生抑制作用。例如,c=a and b 中,如果 a 为假值,那么 c 就为假值,测试程序就不检测 b 了,b 的正确与否就无法测试了。同样,c=a or b 中,如果 a 为真值,那么 c 就为真值,测试程序也不检测 b 了,b 的正确与否也就无法测试了。

表 4-6(a)　测试用例组 6

测试用例	x,y,z	(x>100)	(y>500)	(x>=1000)	(z>5000)	执行路径
Test Case 1	2000,600,6000	True	True	True	True	ace
Test Case 8	50,200,2000	False	False	False	False	abd

表 4-6(b)　测试用例组 6

测试用例	x,y,z	(x>100)and (y>500)	(x>=1000)or (z>5000)	执行路径
Test Case 1	2000,600,6000	True	True	ace
Test Case 8	50,200,2000	False	False	abd

4.1.5　条件组合覆盖

条件组合覆盖是指设计若干个测试用例,执行被测试程序时,程序中每个判断条件的

内部判断式的各种真假组合可能都至少被执行一遍。可见,满足条件组合覆盖的测试用例组一定满足判断覆盖、条件覆盖和判断/条件覆盖。

测试用例组 7:

Test Case 1: x=2000, y=600, z=2000

Test Case 6: x=2000, y=200, z=6000

Test Case 7: x=2000, y=600, z=2000

Test Case 8: x=50, y=200, z=2000

如表 4-7(a)和表 4-7(b)所示,其中表 4-7(a)表示每个判断条件的每个判断式的真值和假值,表 4-7(b)表示每个判断条件的真值和假值。测试用例组 7 虽然满足了判断覆盖、条件覆盖以及判断/条件覆盖,但是并没有覆盖程序控制流图中全部的 4 条路径(ace,abe,abe,abd),只覆盖了其中 3 条路径(ace,abe,abd)。软件测试的目的是尽可能地发现所有软件缺陷,因此程序中的每一条路径都应该进行相应的覆盖测试,从而保证程序中的每一个特定的路径方案都能顺利运行,能够达到这样要求的是路径覆盖测试。

表 4-7(a) 测试用例组 7

测试用例	x,y,z	(x>100)	(y>500)	(x>=1000)	(z>5000)	执行路径
Test Case 1	2000,600,6000	True	True	True	True	ace
Test Case 6	50,600,6000	False	True	False	True	abe
Test Case 7	2000,200,1000	True	False	True	False	abe
Test Case 8	50,200,2000	False	False	False	False	abd

表 4-7(b) 测试用例组 7

测试用例	x,y,z	(x>100)and (y>500)	(x>=1000)or (z>5000)	执行路径
Test Case 1	2000,600,6000	True	True	ace
Test Case 6	50,600,6000	False	True	abe
Test Case 7	2000,200,1000	False	True	abe
Test Case 8	50,200,2000	False	False	abd

4.1.6 路径覆盖

路径覆盖(Path Coverage)要求设计若干测试用例,执行被测试程序时,能够覆盖程序中所有的可能路径。

测试用例组 8:

Test Case 1: x=2000,y=600,z=6000

Test Case 3: x=50,y=600,z=2000

Test Case 4: x=2000,y=600,z=2000

Test Case 7：x＝2000，y＝200，z＝1000

如表 4-8(a)和表 4-8(b)所示，其中表 4-8(a)表示每个判断条件的每个判断式的真值和假值，表 4-8(b)表示每个判断条件的真值和假值。测试用例组 8 可以达到路径覆盖。

表 4-8(a)　测试用例组 8

测试用例	x,y,z	(x>100)	(y>500)	(x>=1000)	(z>5000)	执行路径
Test Case 1	2000,600,6000	True	True	True	True	ace
Test Case 3	50,600,2000	False	True	False	False	abd
Test Case 4	2000,600,2000	True	True	True	False	acd
Test Case 7	2000,200,1000	True	False	True	False	abe

表 4-8(b)　测试用例组 8

测试用例	x,y,z	(x>100)and (y>500)	(x>=1000)or (z>5000)	执行路径
Test Case 1	2000,600,6000	True	True	ace
Test Case 3	50,600,2000	False	False	abd
Test Case 4	2000,600,2000	True	True	acd
Test Case 7	2000,200,1000	False	True	abe

应该注意的是，上面 6 种覆盖测试方法所引用的公共程序只有短短 4 行，是一段非常简单的示例代码。然而在实际测试程序中，一个简短的程序，其路径数目是一个庞大的数字。要对其实现路径覆盖测试是很难的。所以，路径覆盖测试是相对的，要尽可能把路径数压缩到一个可承受范围。

当然，即便对某个简短的程序段做到了路径覆盖测试，也不能保证源代码不存在其他软件问题了。其他的软件测试手段也必要的，它们之间是相辅相成的。没有一个测试方法能够找尽所有软件缺陷，只能说是尽可能多地查找软件缺陷。

4.2　路径分析测试

着眼于路径分析的测试称为路径分析测试。完成路径测试的理想情况是做到路径覆盖。路径覆盖也是白盒测试最为典型的问题。独立路径选择和 Z 路径覆盖是两种常见的路径覆盖方法。

4.2.1　控制流图

白盒测试是针对软件产品内部逻辑结构进行测试的，测试人员必须对测试中的软件有深入的理解，包括其内部结构、各单元部分及之间的内在联系，还有程序运行原理等。因而这是一项庞大并且复杂的工作。为了更加突出程序的内部结构，便于测试人员理解

源代码，可以对程序流程图进行简化，生成控制流图（Control Flow Graph）。简化后的控制流图是由结点和控制边组成的。

1. 控制流图的特点

控制流图有以下几个特点：

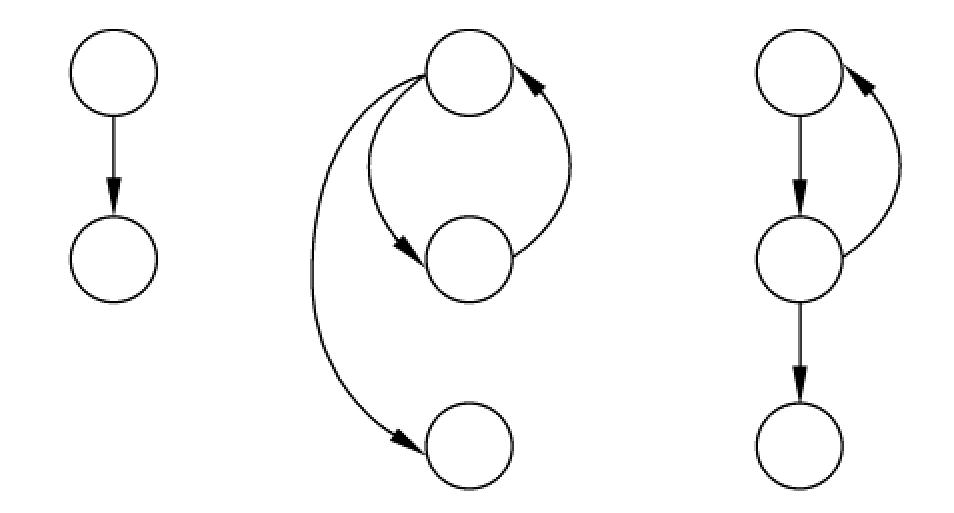
（1）具有惟一入口结点，即源结点，表示程序段的开始语句；

（2）具有惟一出口结点，即汇结点，表示程序段的结束语句；

（3）结点由带有标号的圆圈表示，表示一个或多个无分支的源程序语句；

（4）控制边由带箭头的直线或弧表示，代表控制流的方向。

常见的控制流图如图 4-2 所示。



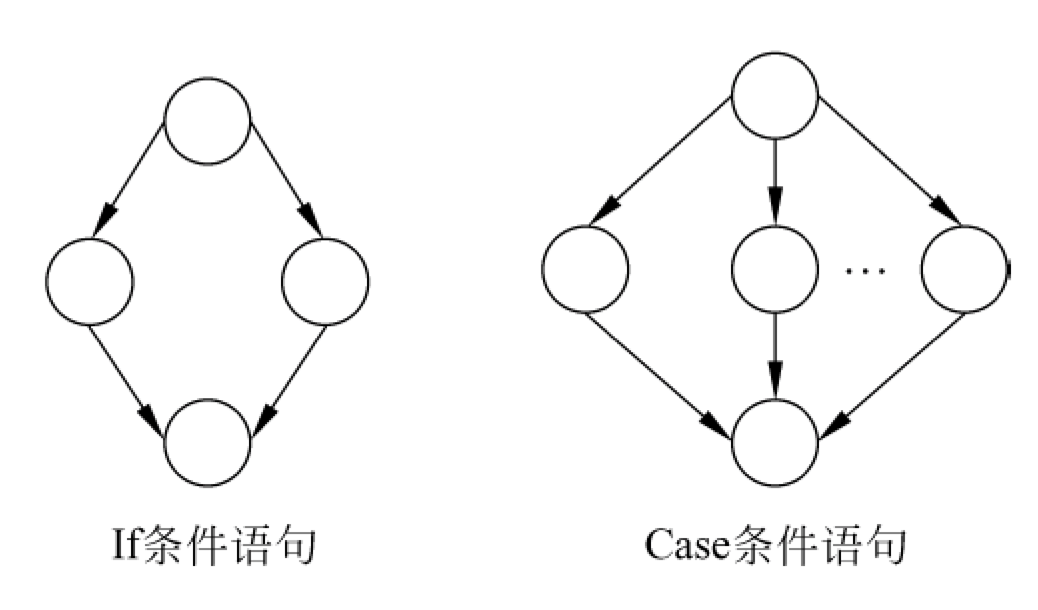


图 4-2　常见的控制流图

包含条件的结点被称为判断结点，由判断结点发出的边必须终止于某一个结点。

2. 程序环路复杂性

程序的环路复杂性是一种描述程序逻辑复杂度的标准，该标准运用基本路径方法，给出了程序基本路径集中的独立路径条数，这是确保程序中每个可执行语句至少执行一次所必需的测试用例数目的上界。

给定一个控制流图 G，设其环形复杂度为 V(G)，在这里介绍三种常见的计算方法来求解 V(G)。

（1）$V(G)=E-N+2$，其中 E 是控制流图 G 中边的数量，N 是控制流图中结点的数目。

（2）$V(G)=P+1$，其中 P 是控制流图 G 中判断结点的数目。

(3) V(G)=A,其中 A 是控制流图 G 中区域的数目。由边和结点围成的区域叫做区域,当在控制流图中计算区域的数目时,控制流图外的区域也应记为一个区域。

4.2.2 独立路径测试

从前面学过的 4.1 节逻辑覆盖测试中可知,对于一个较为复杂的程序要做到完全的路径覆盖测试是不可能实现的。既然路径覆盖测试无法达到,那么可以对某个程序的所有独立路径进行测试,也就是说检验了程序的每一条语句,从而达到语句覆盖,这种测试方法就是独立路径测试方法。从控制流图来看,一条独立路径是至少包含有一条在其他独立路径中从未有过的边的路径。路径可以用控制流图中的结点序列来表示。

例如,在如图 4-3 所示的控制流图中,一组独立的路径是

path 1:1 → 11

path 2:1 → 2 → 3 → 4 → 5 → 10 → 1 → 11

path 3:1 → 2 → 3 → 6 → 8 → 9 → 10 → 1 → 11

path 4:1 → 2 → 3 → 6 → 7 → 9 → 10 → 1 → 11

路径 path 1,path 2,path 3,path 4 组成了控制流图的一个基本路径集。

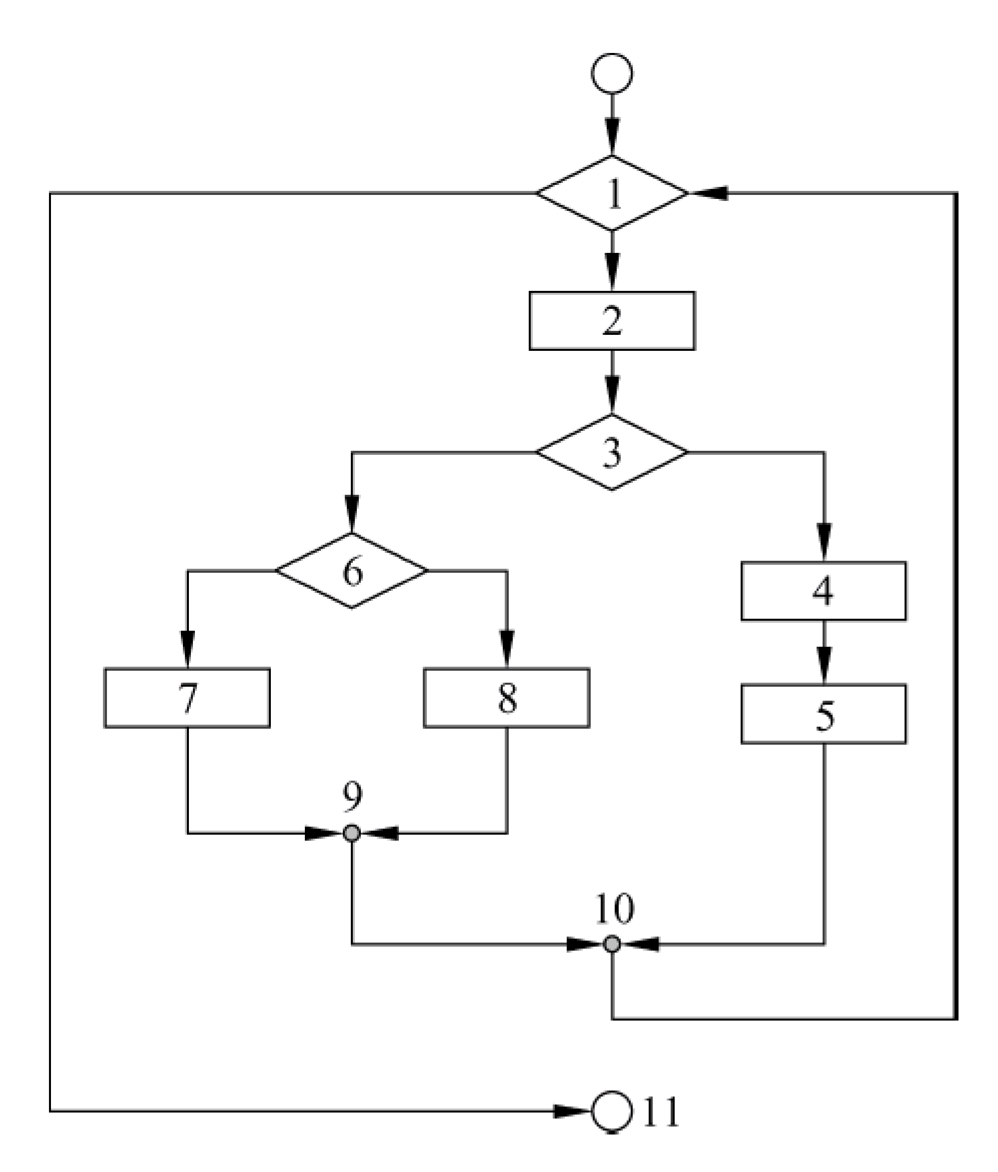


图 4-3 控制流图示例

白盒测试可以设计成基本路径集的执行过程。通常,基本路径集并不惟一确定。

独立路径测试的步骤包括 3 个方面:

- 导出程序控制流图
- 求出程序环形复杂度
- 设计测试用例(Test Case)

下面通过一个 C 语言程序实例来具体说明独立路径测试的设计流程。这段程序是统计一行字符中有多少个单词，单词之间用空格分隔开。

程序 4-2：

```
1   main ()
2   {
3    int num1=0, num2=0, score=100;
4    int i;
5    char str;
6    scanf ("%d, %c\n", &i, &str);
7    while (i<5)
8    {
9       if (str='T')
10           num1++;
11     else if (str='F')
12     {
13        score=score-10;
14        num2++;
15      }
16     i++;
17   }
18   printf ("num1=%d, num2=%d, score=%d\n", num1, num2, score);
19   }
```

1. 导出程序控制流图

根据源代码可以导出程序的控制流图，如图 4-4 所示。每个圆圈代表控制流图的结点，可以表示一个或多个语句。圆圈中的数字对应程序中某一行的编号。箭头代表边的方向，即控制流方向。

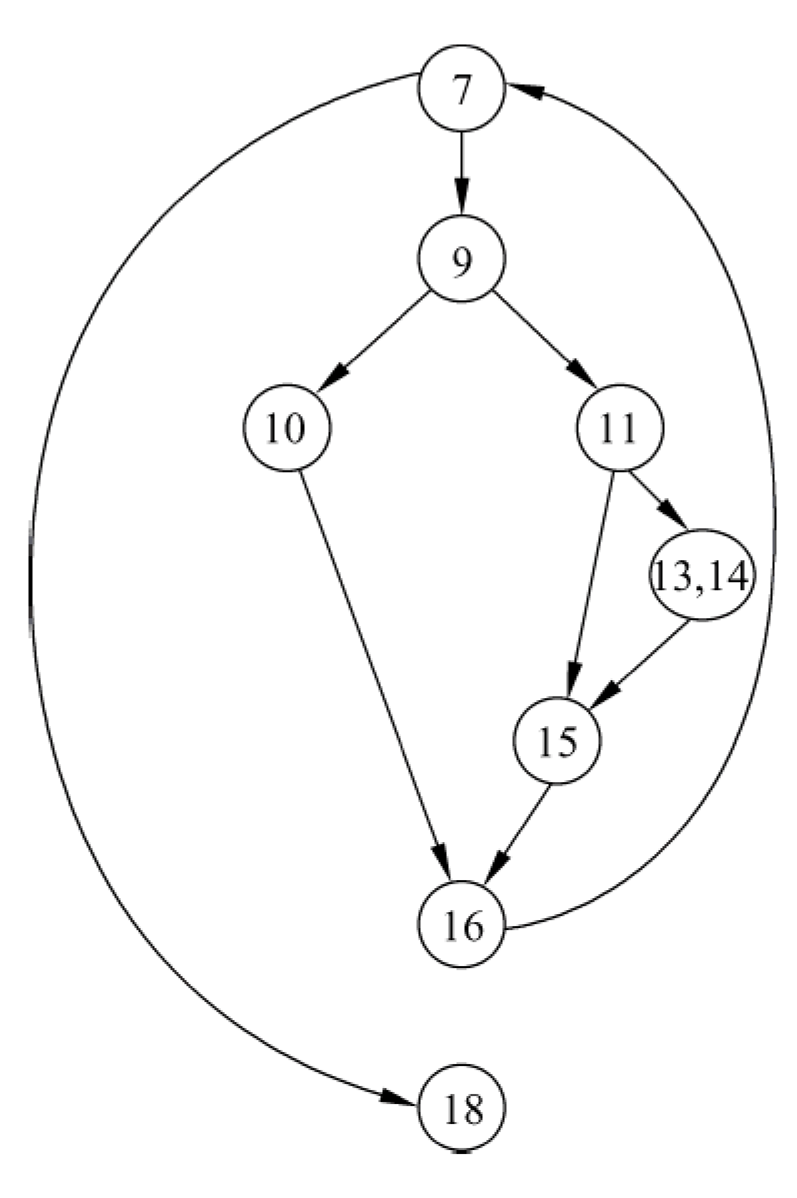


图 4-4　程序 4-2 的控制流图

2. 求出程序环形复杂度

根据程序环形复杂度的计算公式，求出程序路径集合中的独立路径数目。

公式 1：V(G)=10-8+2，其中 10 是控制流图 G 中边的数量，8 是控制流图中结点的数目。

公式 2：V(G)=3+1，其中 3 是控制流图 G 中判断结点的数目。

公式 3：V(G)=4，其中 4 是控制流图 G 中区域的

数目。

因此,控制流图 G 的环形复杂度是 4。就是说至少需要 4 条独立路径组成基本路径集合,并由此得到能够覆盖所有程序语句的测试用例。

3. 设计测试用例

根据上面环形复杂度的计算结果,源程序的基本路径集合中有 4 条独立路径:

path 1: 7→18

path 2: 7→9→10→16→7→18

path 3: 7→9→11→15→16→7→18

path 4: 7→9→11→13→14→15→16→7→18

根据上述 4 条独立路径,设计了测试用例组 9,如表 4-9 所示。测试用例组 9 中的 4 个测试用例作为程序输入数据,能够遍历这 4 条独立路径。对于源程序中的循环体,测试用例组 9 中的输入数据使其执行零次或一次。

表 4-9 测试用例组 9

测试用例	输入		期望输出			执行路径
	i	str	num1	num2	score	
Test Case 1	5	'T'	0	0	100	路径 1
Test Case 2	4	'T'	1	0	100	路径 2
Test Case 3	4	'A'	0	0	100	路径 3
Test Case 4	4	'F'	0	1	90	路径 4

注意:如果程序中的条件判断表达式是由一个或多个逻辑运算符(and,or,not)连接的复合条件表达式,则需要变换为一系列只有单个条件的嵌套的判断。

程序 4-3:

```
1  if (a or b)
2  then
3     procedure x
4  else
5     procedure y;
6  …
```

对应的控制流图如图 4-5 所示,程序行 1 的 a,b 都是独立的判断结点,还有程序行 4 也是判断结点,所以共计 3 个判断结点。图 4-5 的环形复杂度为 V(G)=3+1,其中 3 是图 4-5 中判断结点的数目。

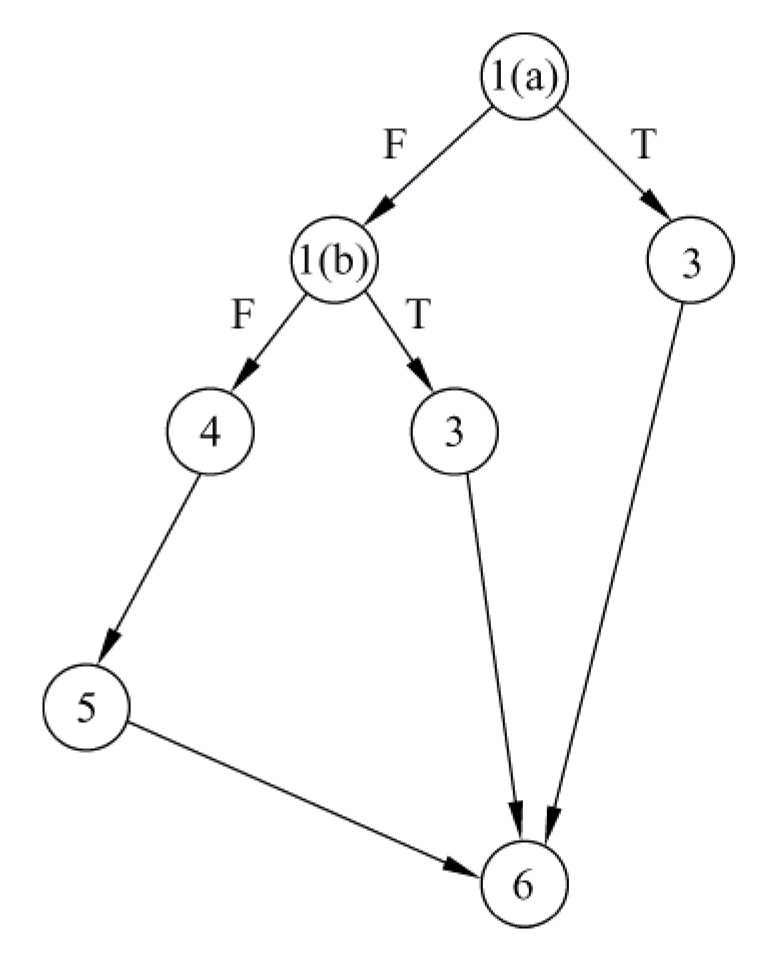


图 4-5 程序 4-3 的控制流图

4.2.3　Z 路径覆盖测试

和独立路径选择一样，Z 路径覆盖也是一种常见的路径覆盖方法。可以说 Z 路径覆盖是路径覆盖面的一种变体。对于语句较少的简单程序，路径覆盖是具有可行性的。但是对于源代码很多的复杂程序，或者对于含有较多条件语句和较多循环体的程序来说，需要测试的路径数目会成倍增长，达到一个巨大数字，以至于无法实现路径覆盖。

为了解决这一问题，必须舍弃一些不重要的因素，简化循环结构，从而极大地减少路径的数量，使得覆盖这些有限的路径成为可能。采用简化循环方法的路径覆盖就是 Z 路径覆盖。所谓简化循环就是减少循环的次数。不考虑循环体的形式和复杂度如何，也不考虑循环体实际上需要执行多少次，只考虑通过循环体零次和一次这两种情况。这里的零次循环是指跳过循环体，从循环体的入口直接到循环体的出口。通过一次循环体是指检查循环初始值。

根据简化循环的思路，循环要么执行，要么跳过，这和判定分支的效果是一样的。可见，简化循环就是将循环结构转变成选择结构。

如图 4-6(a)和图 4-6(b)所示表示了两种最典型的循环控制结构。图 4-6(a)是先比较循环条件后执行循环体，循环体 B 可能执行也可能不被执行。假设限定循环体 B 执行零次和一次，这样就和图 4-6(c)的条件结构一样了。图 4-6(b)是先执行循环体后比较循环条件。假设循环体 B 被执行一次，再经过条件判断跳出循环，那么其效果就和图 4-6(c)的条件结构只执行右分支的效果一样了。

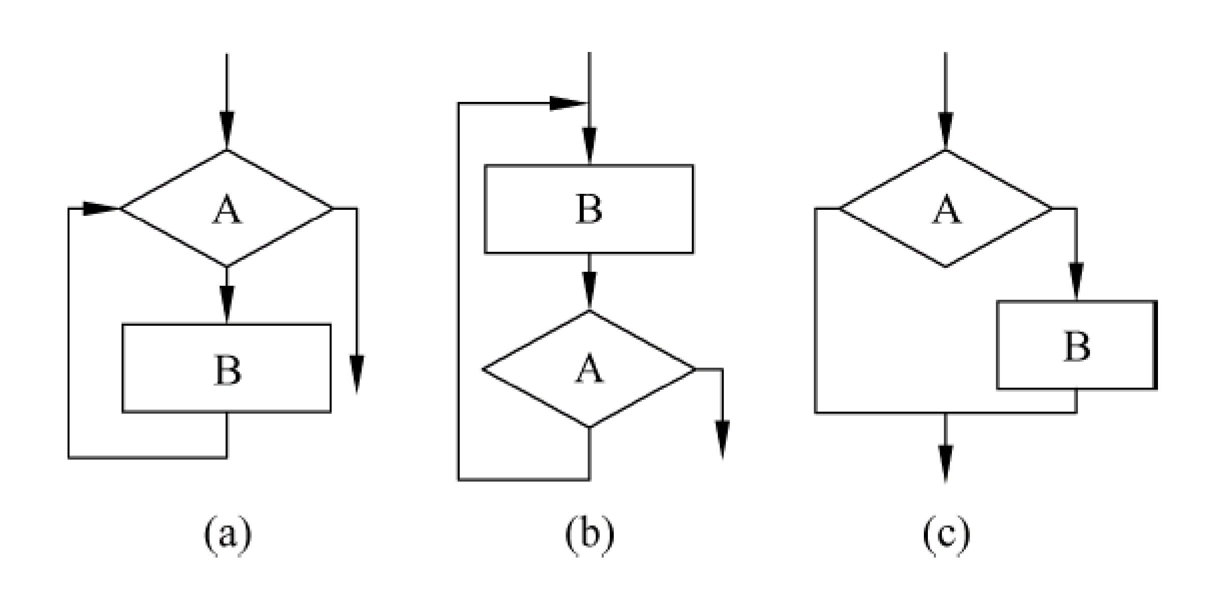


图 4-6　循环结构和条件结构

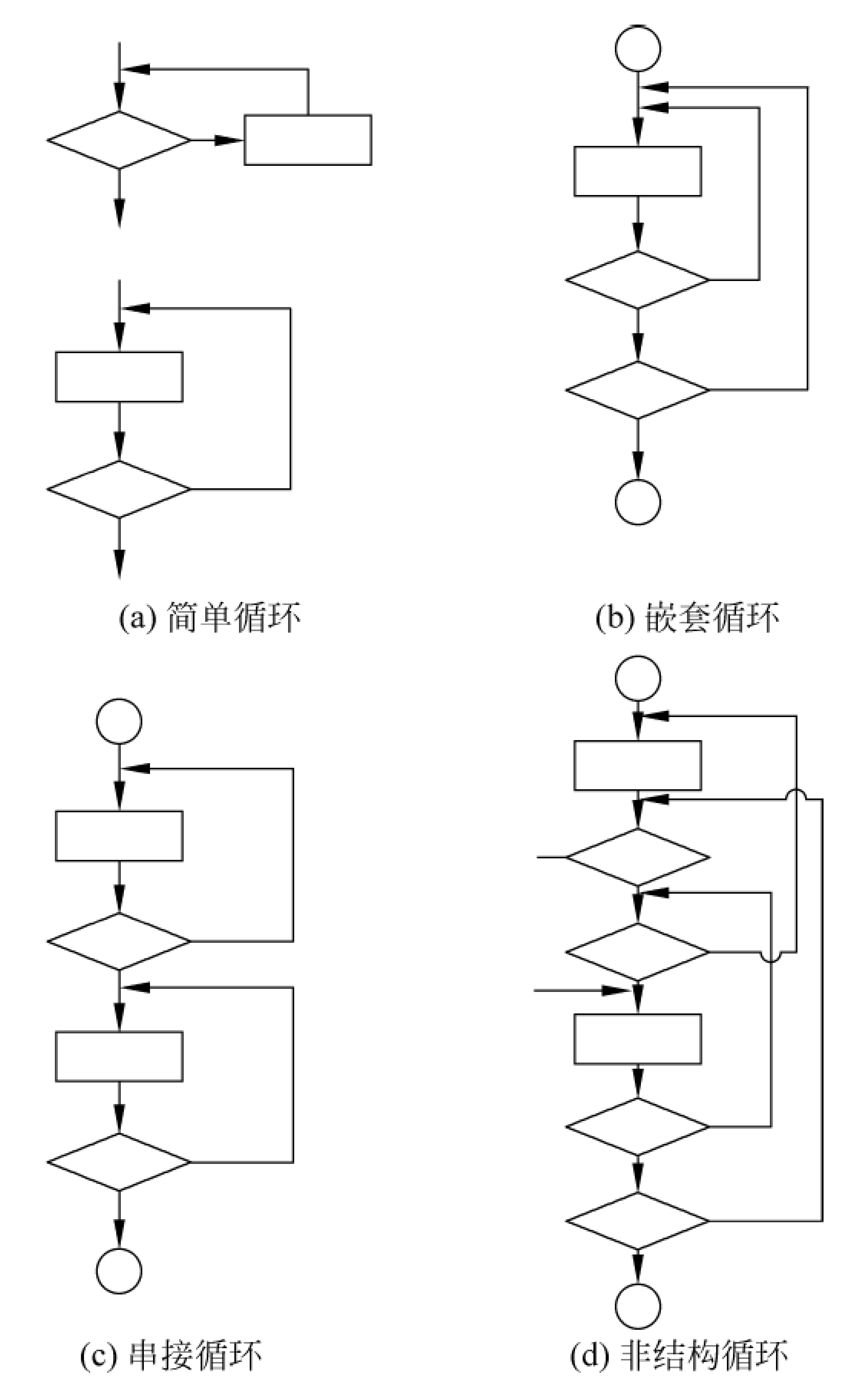
一旦将循环结构简化为选择结构后，路径的数量将大大减少，这样就可以实现路径覆盖测试了。对于实现简化循环的程序，可以将程序用路径树来表示。当得到某一程序的路径树后，从其根结点开始，一次遍历，再回到根结点时，将所经历的叶子结点名排列起来，就得到一个路径。如果已经遍历了所有叶子结点，那就得到了所有的路径。当得到所有的路径后，生成每个路径的测试用例，就可以实现 Z 路径覆盖测试。

4.3 其他白盒测试方法

白盒测试除了覆盖测试和路径分析测试两大类方法之外，还有很多其他常见的测试方法，如循环测试、变异测试、程序插装等。这些方法相辅相成，可以增强测试效果，提高测试效率。

4.3.1 循环测试

循环测试是一种着重循环结构有效性测试的白盒测试方法。循环结构测试用例的设计有以下 4 种模式，分别如图 4-7(a)、图 4-7(b)、图 4-7(c)、图 4-7(d)所示。



(a) 简单循环　(b) 嵌套循环

(c) 串接循环　(d) 非结构循环

图 4-7　循环测试的模式

1. 简单循环

对于图 4-7(a)所示的简单循环，设计测试用例时，有以下几种测试集情况，其中 n 是

可以通过循环体的最大次数：

(1) 零次循环：跳过循环体，从循环入口到出口；

(2) 通过一次循环体：检查循环初始值；

(3) 通过两次循环体：检查两次循环；

(4) m 次通过循环体($m<n$)：检查多次循环；

(5) n，$n-1$，$n+1$ 次通过循环体：检查最大次数循环以及比最大次数多一次、少一次的循环。

2. 嵌套循环

对于图 4-7(b)所示的嵌套循环，如果采用简单循环中的测试集来测试嵌套循环，可能的测试数目就会随着嵌套层数的增加成几何级地增长。这样的测试是无法实现的。所以，要减少测试数目。

(1) 对最内层循环按照简单循环的测试方法进行测试，把其他外层循环设置为最小值；

(2) 逐步外推，对其外面一层的循环进行测试。测试时保持本次循环的所有外层循环仍取最小值，而由本层循环嵌套的循环取某些"典型"值；

(3) 反复进行(2)中操作，向外层循环推进，直到所有各层循环测试完毕。

3. 串接循环

对于图 4-7(c)所示的串接循环，如果串接循环的循环体之间是彼此独立的，那么可以采用简单循环的测试方法进行测试。如果串接循环的循环体之间有关联，例如前一个循环体的结果是后一个循环体的初始值，那么需要应用嵌套循环的测试方法进行测试。

4. 非结构循环

对于图 4-7(d)所示的非结构循环，不能进行测试，需要重新设计出结构化的程序后再进行测试。

4.3.2 变异测试

变异测试是一种故障驱动测试，即针对某一类特定程序故障进行的测试，变异测试也是一种比较成熟的排错性测试方法。它可以通过检验测试数据集的排错能力来判断软件测试的充分性。

那么程序变异以及变异测试到底是什么呢？

假设对程序 P 进行一些微小改动而得到程序 MP，程序 MP 就是程序 P 的一个变异体。假设程序 P 在测试集 T 上是正确的，设计某一变异体集合：M={MP|MP 是 P 的变异体}，若变异体集合 M 中的每一个元素在 T 上都存在错误，则认为源程序 P 的正确度较高，否则若 M 中的某些元素在 T 上运行正确，则可能存在以下一些情况：

- M 中的这些变异体在功能上与源程序 P 是等价的；
- 现有的测试数据不足以找出源程序 P 与其变异体之间的差别；
- 源程序 P 可能产生故障，而其某些变异体却是正确的。

可见，测试集 T 和变异体集合 M 中的每一个变异体 MP 的选择都是很重要的，它们会直接影响变异测试的测试效果。

那么如何建立变异体呢？变异体是变异运算作用在源程序上的结果。被测试的源程序经过变异运算会产生一系列不同的变异体。例如，将数据元素用其他数据元素替代，将常量值增加或减少，改动数组分量，变换操作符，替换或删除某些语句等。

总之，对程序进行变换的方法多种多样，具体操作要靠测试人员的实际经验。通过变异分析构造测试数据集的过程是一个循环过程，当对源程序及其变异体进行测试后，若发现某些变异体并不理想，就要适当增加测试数据，直到所有变异体达到理想状态，即变异体集合中的每一个变异体在 T 上都存在错误。

4.3.3 程序插装

程序插装是借助于在被测程序中设置断点或打印语句来进行测试的方法，在执行测试的过程中可以了解一些程序的动态信息。这样在运行程序时，既能检验测试的结果数据，又能借助插入语句给出的信息掌握程序的动态运行特性，从而把程序执行过程中所发生的重要事件记录下来。

程序插装设计时主要考虑三方面因素：

- 需要探测哪些信息；
- 在程序的什么位置设立插装点；
- 计划设置多少个插装点。

插装技术在软件测试中主要有以下几个应用：

(1) 覆盖分析

程序插装可以估计程序控制流图中被覆盖的程度，确定测试执行的充分性，从而设计更好的测试用例，提高测试覆盖率。

(2) 监控

在程序的特定位置设立插装点，插入用于记录动态特性的语句，用来监控程序运行时的某些特性，从而排除软件故障。

(3) 查找数据流异常

程序插装可以记录在程序执行中某些变量值的变化情况和变化范围。掌握了数据变量的取值状况，就能准确地判断是否发生数据流异常。虽然数据流异常可以用静态分析器来发现，但是使用插装技术可以更经济更简便，毕竟所有信息的获取是随着测试过程附带得到的。

4.4　实例设计

实例 1　运用逻辑覆盖的方法测试程序

程序 4-4：

```
1  If (x>1&& y=1) then
2    z=z*2
3  If (x=3|| z>1) then
4    y++;
```

运用逻辑覆盖的方法设计测试用例组，如表 4-10 所示。

表 4-10　测试用例组 10

逻辑覆盖方法	测试用例组	执行路径
语句覆盖	x=3,y=1,z=2	1,2,3,4
判断覆盖	x=3,y=1,z=2	1,2,3,4
	x=1,y=1,z=1	1,3
条件覆盖	x=3,y=0,z=1	1,3,4
	x=1,y=1,z=2	1,3,4
判断/条件覆盖	x=3,y=1,z=2	1,2,3,4
	x=1,y=0,z=1	1,3
条件组合覆盖	x=3,y=1,z=2	1,2,3,4
	x=3,y=0,z=1	1,3,4
	x=1,y=1,z=2	1,3,4
	x=1,y=0,z=1	1,3
路径覆盖	x=3,y=1,z=2	1,2,3,4
	x=3,y=0,z=1	1,3,4
	x=2,y=1,z=1	1,2,3
	x=1,y=1,z=1	1,3

实例 2　运用路径分析的方法测试程序

程序 4-5：

```
main ()
{
  int flag, t1, t2, a=0, b=0;
  scanf ("%d, %d, %d\n", &flag, &t1, &t2);
  while (flag>0)
  {
```

```
    a=a+1;
    if (t1=1)
    then
    {
        b=b+1;
        flag=0;
     }
    else
    {
        if (t2=1)
        then b=b-1;
        else a=a-2;
        flag-- ;
     }
    }
    printf("a=%d, b=d%\n", a, b);
}
```

程序 4-5 的流程图如图 4-8 所示。

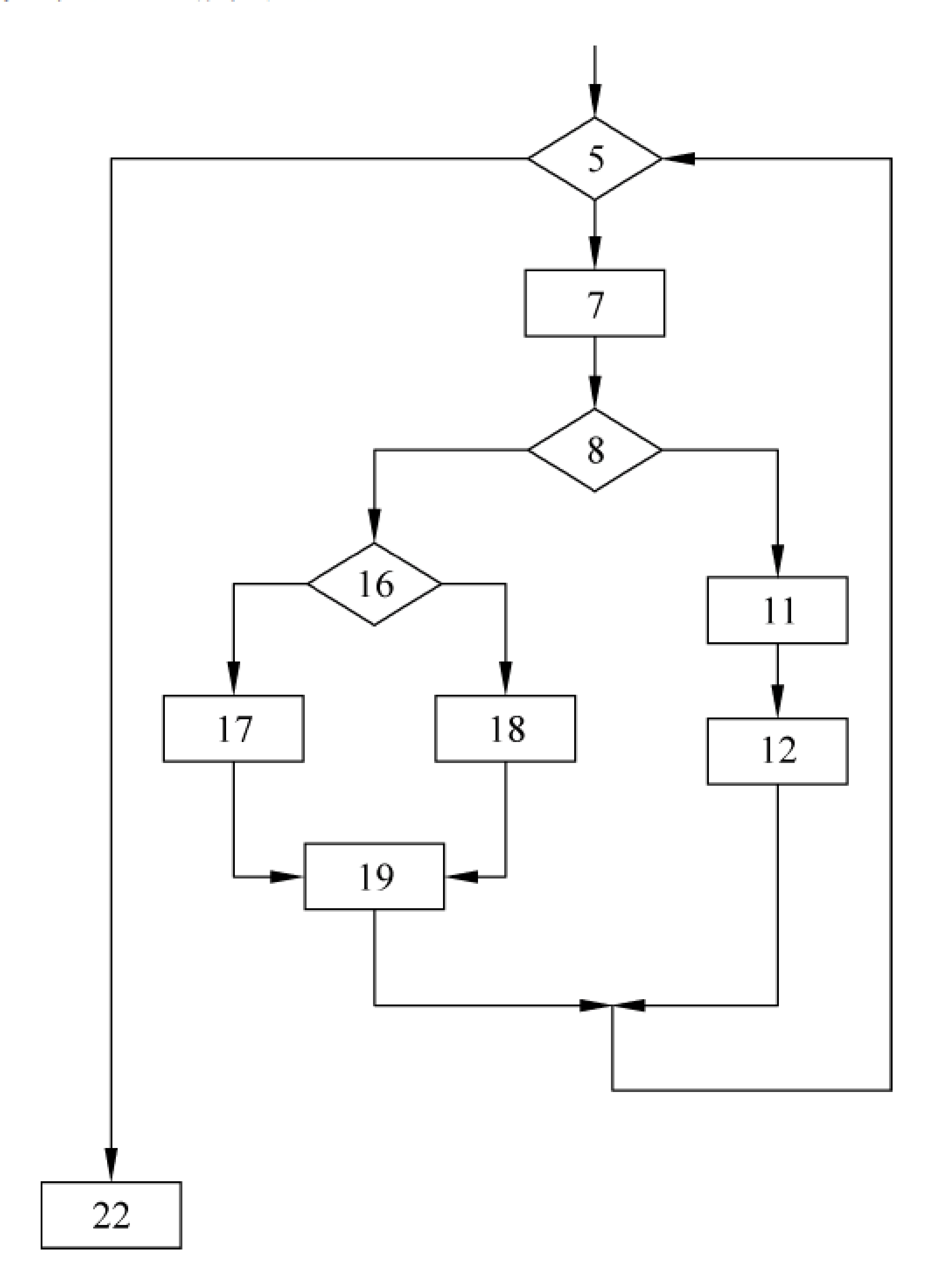


图 4-8　程序 4-5 的流程图

程序 4-5 的控制流图如图 4-9 所示，其中 R1、R2、R3 和 R4 代表控制流图的 4 个区域。R4 代表的是控制流图外的区域，也算作控制流图的一个区域。

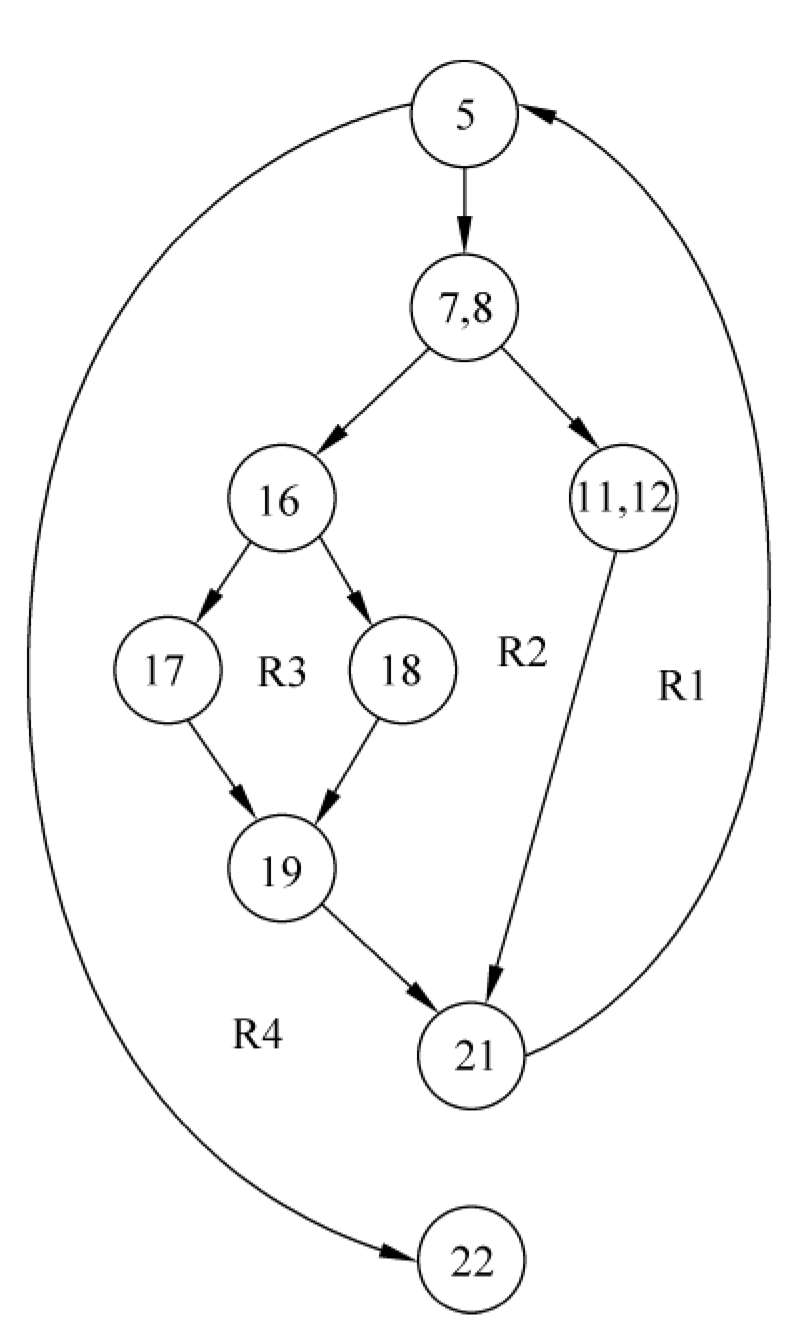


图 4-9　程序 4-5 的控制流图

下面运用路径分析的方法设计测试用例组。

(1) 根据程序环形复杂度的计算公式，求出程序路径集合中的独立路径数目。

公式 1：V(G)＝11－9＋2，其中 10 是控制流图 G 中边的数量，8 是控制流图中结点的数目。

公式 2：V(G)＝3＋1，其中 3 是控制流图 G 中判断结点的数目。

公式 3：V(G)＝4，其中 4 是控制流图 G 中区域的数目。

因此，控制流图 G 的环形复杂度是 4。

(2) 根据上面环形复杂度的计算结果，源程序的基本路径集合中有 4 条独立路径：

path 1：5→22

path 2：5→7，8→11，12→21→5→22

path 3：5→7，8→16→17→19→21→5→22

path 4：5→7，8→16→18→19→21→5→22

(3) 设计测试用例组 11 如表 4-11 所示。根据上述 4 条独立路径设计出了这组测试用例，其中的 4 组数据能够遍历各个独立路径，也就满足了路径分析测试的要求。

需要注意的是，对于源程序中的循环体，测试用例组 11 中的输入数据使其执行零次或一次。

表 4-11　测试用例组 11

测试用例	输　入			期望输出		执行路径
	flag	t1	t2	a	b	
Test Case 1	0	1	1	0	0	路径 1
Test Case 2	1	1	0	1	1	路径 2
Test Case 3	1	0	1	1	－1	路径 3
Test Case 4	1	0	0	－1	0	路径 4

小　结

白盒测试是基于被测程序的源代码设计测试用例的测试方法。常见的白盒测试方法有逻辑覆盖测试和路径分析测试两大类。

在逻辑覆盖测试中，按照覆盖策略由弱到强的严格程度，介绍了语句覆盖、判断覆盖、条件覆盖、判断/条件覆盖、条件组合覆盖和路径覆盖 6 种覆盖测策略。

- 语句覆盖：每个语句至少执行一次。
- 判定覆盖：在语句覆盖的基础上，每个判定的每个分支至少执行一次。
- 条件覆盖：在语句覆盖的基础上，使每个判定表达式的每个条件都取到各种可能的结果。
- 判定/条件覆盖：即判定覆盖和条件覆盖的交集。
- 条件组合覆盖：每个判定表达式中条件的各种可能组合都至少出现一次。
- 路径覆盖：每条可能的路径都至少执行一次，若图中有环，则每个环至少经过一次。

在路径分析测试中，介绍了独立路径测试和 Z 路径覆盖测试两种常用方法。

- 独立路径测试方法把覆盖的路径数压缩到一定限度内，程序中的循环体最多只执行一次，对程序中所有独立路径进行测试。它是在程序控制流图的基础上，分析控制构造的环路复杂性，导出基本可执行路径集合，设计测试用例的方法。设计出的测试用例要保证程序的每一个可执行语句至少要执行一次。
- Z 路径覆盖测试是指采用简化循环的方法进行路径覆盖测试。被测源程序中的循环体执行零次或一次。

最后，介绍了其他一些白盒测试方法。循环测试是一种着重循环结构有效性测试的测试方法。变异测试是一种故障驱动测试，是针对某一类特定程序故障进行的测试。程序插装是借助于在被测程序中设置断点或打印语句来进行测试的方法，在执行测试的过程中可以了解一些程序的动态信息。

习　题

1. 阐述白盒测试的各种方法，进行分析总结。
2. 分析归纳逻辑覆盖测试的 6 种覆盖策略各自的特点。
3. 简述独立路径测试的基本步骤。

4. 对下列 C 语言程序设计逻辑覆盖测试用例。

```
Void test(int X, int A, int B)
 {
If (A>1&& B=0) then
  X=X/A
If (A=2|| X>1) then
  X=X+1;
}
```

第 5 章

软件测试计划与文档

本章概述

软件测试的目的是尽可能早一些找出软件缺陷,并确保其得以修复。软件测试人员不断追求着低成本下的高效率测试,而成功的测试要依靠有效的测试计划、测试用例和软件测试报告,它们也是测试过程要解决的核心问题。

本章主要介绍软件测试计划的制定、测试文档的形成、测试用例的设计以及测试报告的编写格式。

5.1 测试计划的制定

5.1.1 测试计划

软件测试是一个有组织有计划的活动,应当给予充分的时间和资源进行测试计划,这样软件测试才能在合理的控制下正常进行。测试计划(Test Planning)作为测试的起始步骤,是整个软件测试过程的关键管理者。

1. 测试计划的定义

测试计划规定了测试各个阶段所要使用的方法策略、测试环境、测试通过或失败的准则等内容。《ANSI/IEEE 软件测试文档标准 829—1983》将测试计划定义为:“一个叙述了预定的测试活动的范围、途径、资源及进度安排的文档。它确认了测试项、被测特征、测试任务、人员安排,以及任何偶发事件的风险。”

2. 测试计划的目的和作用

测试计划的目的是明确测试活动的意图。它规范了软件测试内容、方法和过程,为有组织地完成测试任务提供保障。专业的测试必须以一个好的测试计划作为基础。尽管测试的每一个步骤都是独立的,但是必须有一个起到框架结构作用的测试计划。

3. 测试计划书

测试计划文档化就成为测试计划书，包含总体计划也包含分级计划，是可以更新改进的文档。从文档的角度看，测试计划书是最重要的测试文档，完整细致并具有远见性的计划书会使测试活动安全顺利地向前进行，从而确保所开发的软件产品的高质量。

4. 测试计划的内容

软件测试计划是整个测试过程中最重要的部分，为实现可管理且高质量的测试过程提供基础。测试计划以文档形式描述软件测试预计达到的目标，确定测试过程所要采用的方法策略。测试计划包括测试目的、测试范围、测试对象、测试策略、测试任务、测试用例、资源配置、测试结果分析和度量以及测试风险评估等，测试计划应当足够完整但也不应当太详尽。借助软件测试计划，参与测试的项目成员，尤其是测试管理人员，可以明确测试任务和测试方法，保持测试实施过程的顺畅沟通；跟踪和控制测试进度，应对测试过程中的各种变更。因此一份好的测试计划需要综合考虑各种影响测试的因素。

实际的测试计划内容因不同的测试对象而灵活变化，但通常来说一个正规的测试计划应该包含以下几个项目，也可以看做是通用的测试计划样本以供参考：

- 测试的基本信息：包括测试目的、背景、测试范围等；
- 测试的具体目标：列出软件需要进行测试的部分和不需要进行测试的部分；
- 测试的策略：测试人员采用的测试方法，如回归测试、功能测试、自动测试等；
- 测试的通过标准：测试是否通过了界定标准以及没有通过情况的处理方法；
- 停测标准：给出每个测试阶段停止测试的标准；
- 测试用例：详细描述测试用例，包括测试值、测试操作过程、测试期望值等；
- 测试的基本支持：测试所需的硬件支持、自动测试软件等；
- 部门责任分工：明确所有参与软件管理、开发、测试、技术支持等部门的责任细则；
- 测试人力资源分配：列出测试所需人力资源以及软件测试人员的培训计划；
- 测试进度安排：制定每一个阶段的详细测试进度安排表；
- 风险估计和危机处理：估计测试过程中潜在的风险以及面临危机时的解决办法。

一个理想的测试计划应该体现以下几个特点：

- 在检测主要缺陷方面有一个好的选择；
- 提供绝大部分代码的覆盖率；
- 具有灵活性；
- 易于执行、回归和自动化；
- 定义要执行测试的种类；
- 测试文档明确说明期望的测试结果；
- 当缺陷被发现时提供缺陷核对；

- 明确定义测试目标；
- 明确定义测试策略；
- 明确定义测试通过标准；
- 没有测试冗余；
- 确认测试风险；
- 文档化确定测试的需求；
- 定义可交付的测试件。

软件测试计划是整个软件测试流程工作的基本依据，测试计划中所列条目在实际测试中必须一一执行。在测试的过程中，若发现新的测试用例，就要尽早补充到测试计划中。若预先制定的测试计划项目在实际测试中不适用或无法实现，那么也要尽快对计划进行修改，使计划具有可行性。

5.1.2 测试计划的制定和软件开发与测试的关系

1. 测试计划的制定

测试的计划与控制是整个测试过程中最重要的阶段，它为实现可管理且高质量的测试过程提供基础。这个阶段需要完成的主要工作内容是：拟定测试计划，论证那些在开发过程中难于管理和控制的因素，明确软件产品的最重要部分(风险评估)。

(1) 概要测试计划

概要测试计划是在软件开发初期制定，其内容包括：

① 定义被测试对象和测试目标；

② 确定测试阶段和测试周期的划分；

③ 制定测试人员，软、硬件资源和测试进度等方面的计划；

④ 明确任务与分配及责任划分；

⑤ 规定软件测试方法、测试标准。比如，语句覆盖率达到98%，三级以上的错误改正率达98%等；

⑥ 所有决定不改正的错误都必须经专门的质量评审组织同意；

⑦ 支持环境和测试工具等。

(2) 详细测试计划

详细测试计划是测试者或测试小组的具体的测试实施计划，它规定了测试者负责测试的内容、测试强度和工作进度，是检查测试实际执行情况的重要标准。

详细测试计划的主要内容有：计划进度和实际进度对照表；测试要点；测试策略；尚未解决的问题和障碍。

(3) 制定主要内容

计划进度和实际进度对照表；测试要点；测试策略；尚未解决的问题和障碍。

(4) 制定测试大纲(用例)

测试大纲是软件测试的依据,保证测试功能不被遗漏,并且功能不被重复测试,使得能合理安排测试人员,使得软件测试不依赖于个人。

测试大纲包括:测试项目、测试步骤、测试完成的标准以及测试方式(手动测试或自动测试)。测试大纲不仅是软件开发后期测试的依据,而且在系统的需求分析阶段也是质量保证的重要文档和依据。无论是自动测试还是手动测试,都必须满足测试大纲的要求。

测试大纲的本质:从测试的角度对被测对象的功能和各种特性的细化和展开。针对系统功能的测试大纲是基于软件质量保证人员对系统需求规格说明书中有关系统功能定义的理解,将其逐一细化展开后编制而成的。

测试大纲的好处:保证测试功能不被遗漏,使得软件功能不被重复测试,合理安排测试人员,使得软件测试不依赖于个人。测试大纲不仅是软件开发后期测试的依据,而且在系统的需求分析阶段也是质量保证的重要文档和依据。

(5) 制定测试通过或失败的标准

测试标准为可观的陈述,它指明了判断/确认测试在何时结束,以及所测试的应用程序的质量。测试标准可以是一系列的陈述或对另一文档(如测试过程指南或测试标准)的引用。

测试标准应该指明:

- 确切的测试目标;
- 度量的尺度如何建立;
- 使用了哪些标准对度量进行评价。

(6) 制定测试挂起标准和恢复的必要条件

指明挂起全部或部分测试项的标准,并指明恢复测试的标准及其必须重复的测试活动。

(7) 制定测试任务安排

明确测试任务,对每项任务都必须明确 7 个主题。

- 任务:用简洁的句子对任务加以说明;
- 方法和标准:指明执行该任务时,应该采用的方法以及所应遵守的标准;
- 输入输出:给出该任务所必需的输入输出;
- 时间安排:给出任务的起始和持续时间;
- 资源:给出任务所需要的人力和物力资源;
- 风险和假设:指明启动该任务应满足的假设,以及任务执行可能存在的风险;
- 角色和职责:指明由谁负责该任务的组织和执行,以及谁将担负怎样的职责。

(8) 制定应交付的测试工作产品

指明应交付的文档、测试代码和测试工具,一般包括的文档有:测试计划、测试方案、测试用例、测试规程、测试日志、测试总结报告、测试输入与输出数据、测试工具。

（9）制定工作量估计

给出前面定义任务的人力需求和总计。

（10）编写测试方案文档

测试方案文档是设计测试阶段的文档，指明为完成软件或软件集成的特性测试而进行的设计测试方法的细节文档。

2. 软件开发、软件测试与测试计划的关系

软件开发、软件测试与测试计划制定的并行关系如图 5-1 所示。

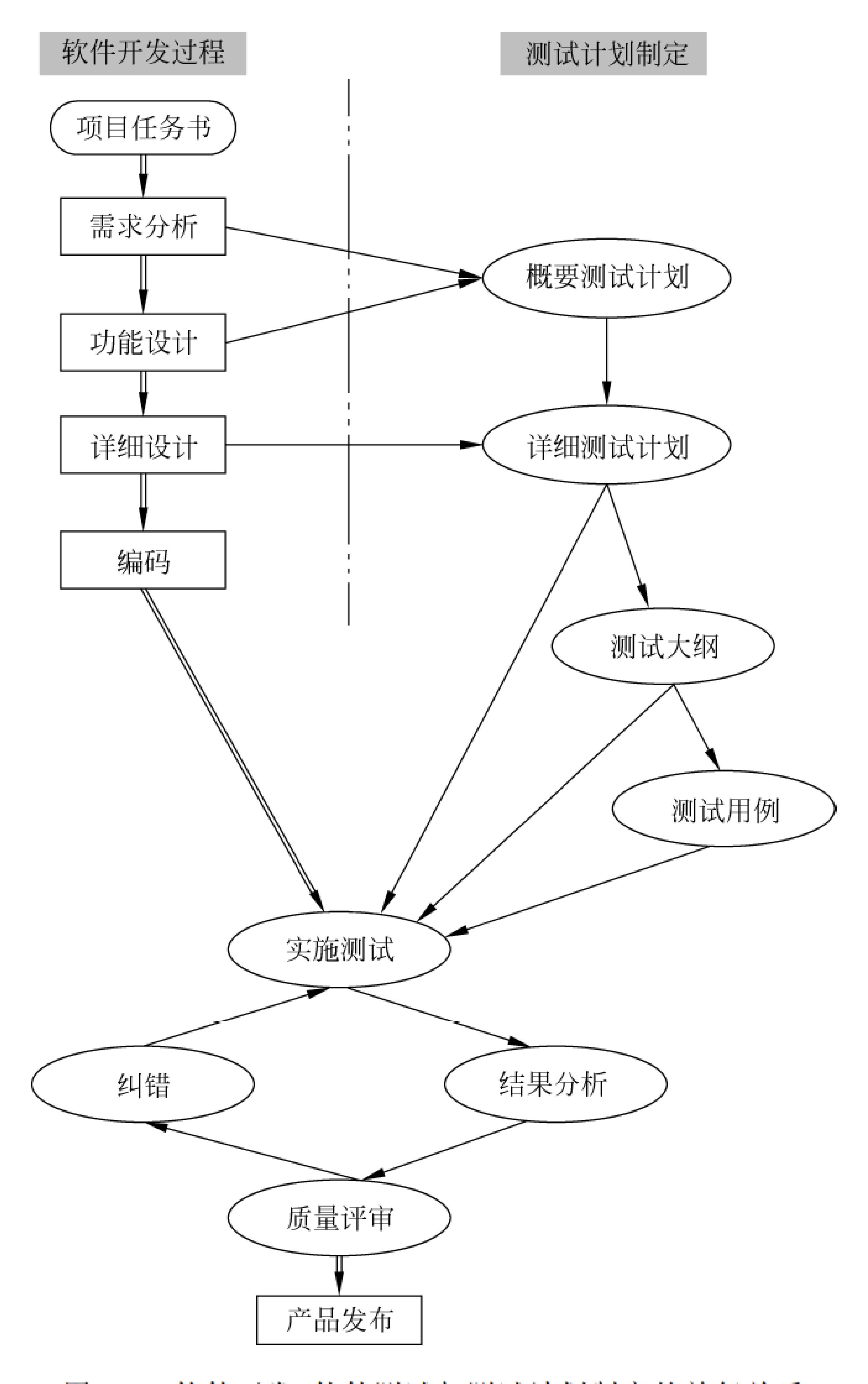


图 5-1　软件开发、软件测试与测试计划制定的并行关系

5.2 测试文档

5.2.1 测试文档的概念

1. 测试文档的定义

测试文档(Testing Documentation)记录和描述了整个测试流程,它是整个测试活动中非常重要的文件。测试过程实施所必备的核心文档是:测试计划、测试用例(大纲)和软件测试报告。

2. 测试文档的重要性

软件测试是一个很复杂的过程,涉及软件开发其他阶段的工作,对于提高软件质量、保证软件正常运行有着十分重要的意义,因此必须把对测试的要求、过程及测试结果以正式的文档形式写下来。软件测试文档用来描述要执行的测试及测试的结果。可以说,测试文档的编制是软件测试工作规范化的一个重要组成部分。

软件测试文档不只在测试阶段才开始考虑,它应在软件开发的需求分析阶段就开始着手编制,软件开发人员的一些设计方案也应在测试文档中得到反映,以利于设计的检验。测试文档对于测试阶段的工作有着非常明显的指导作用和评价作用。即便在软件投入运行的维护阶段,也常常要进行再测试或回归测试,这时仍会用到软件测试文档。

3. 测试文档的内容

整个测试流程会产生很多个测试文档,一般可以把测试文档分为两类:测试计划和测试分析报告。

测试计划文档描述将要进行的测试活动的范围、方法、资源和时间进度等。测试计划中罗列了详细的测试要求,包括测试的目的、内容、方法、步骤以及测试的准则等。在软件的需求和设计阶段就要开始制定测试计划,不能在开始测试的时候才制定测试计划。通常,测试计划的编写要从需求分析阶段开始,直到软件设计阶段结束时才完成。

测试报告是执行测试阶段的测试文档,对测试结果进行分析说明。说明软件经过测试以后,结论性的意见如何,软件的能力如何,存在哪些缺陷和限制等,这些意见既是对软件质量的评价,又是决定该软件能否交付用户使用的依据。由于要反映测试工作的情况,自然应该在测试阶段编写。

测试报告包含了相应的测试项的执行细节。软件测试报告是软件测试过程中最重要的文档,记录问题发生的环境,如各种资源的配置情况,问题的再现步骤以及问题性质的说明。测试报告更重要的是还记录了问题的处理进程,而问题的处理进程从一定角度上反映了测试的进程和被测软件的质量状况以及改善过程。

《计算机软件测试文档编制规范》国家标准给出了更具体的测试文档编制建议，其中包括以下几个内容。

- 测试计划：描述测试活动的范围、方法、资源和进度，其中规定了被测试的对象，被测试的特性、应完成的测试任务、人员职责及风险等。
- 测试设计规格说明：详细描述测试方法，测试用例设计以及测试通过的准则等。
- 测试用例规格说明：测试用例文档描述一个完整的测试用例的必备因素，如输入、预期结果、测试执行条件以及对环境的要求、对测试规程的要求等。测试用例的设计将在5.3节作详细介绍。
- 测试步骤规格说明：测试规格文档指明了测试所执行活动的次序，规定了实施测试的具体步骤。它包括测试规程清单和测试规程列表两部分。
- 测试日志：日志是测试小组对测试过程所作的记录。
- 测试事件报告：报告说明测试中发生的一些重要事件。
- 测试总结报告：对测试活动所作的总结和结论。测试总结报告的格式将在5.4节作具体说明。

上述测试文档中，前4项属于测试计划类文档，后3项属于测试分析报告类文档。

5.2.2 软件生命周期各阶段的测试任务与可交付的文档

通常软件生命周期可分为以下6个阶段：需求阶段、功能设计阶段、详细设计阶段、编码阶段、测试阶段以及运行/维护阶段，相邻两个阶段之间可能存在一定程度的重复以保证阶段之间的顺利衔接，但每个阶段的结束是有一定的标志，例如已经提交可交付文档等。

1. 需求阶段

(1) 测试输入

需求计划(来自开发)。

(2) 测试任务

- 制定验证和确认测试计划；
- 对需求进行分析和审核；
- 分析并设计基于需求的测试，构造对应的需求覆盖或追踪矩阵。

(3) 可交付的文档

- 验证测试计划；
- 验证测试计划(针对需求设计)；
- 验证测试报告(针对需求设计)。

2．功能设计阶段

（1）测试输入

功能设计规格说明（来自开发）。

（2）测试任务

- 功能设计验证和确认测试计划；
- 分析和审核功能设计规格说明；
- 可用性测试设计；
- 分析并设计基于功能的测试，构造对应的功能覆盖矩阵；
- 实施基于需求和基于功能的测试。

（3）可交付的文档

- 确认测试计划；
- 验证测试计划（针对功能设计）；
- 验证测试报告（针对功能设计）。

3．详细设计阶段

（1）测试输入

详细设计规格说明（来自开发）。

（2）测试任务

- 详细设计验证测试计划；
- 分析和审核详细设计规格说明；
- 分析并设计基于内部的测试。

（3）可交付的文档

- 详细确认测试计划；
- 验证测试计划（针对详细设计）；
- 验证测试报告（针对详细设计）；
- 测试设计规格说明。

4．编码阶段

（1）测试输入

代码（来自开发）。

（2）测试任务

- 代码验证测试计划；
- 分析代码；
- 验证代码；
- 设计基于外部的测试；

- 设计基于内部的测试。

（3）可交付的文档

- 测试用例规格说明；
- 需求覆盖或追踪矩阵；
- 功能覆盖矩阵；
- 测试步骤规格说明；
- 验证测试计划（针对代码）；
- 验证测试报告（针对代码）。

5. 测试阶段

（1）测试输入

- 要测试的软件；
- 用户手册。

（2）测试任务

- 制定测试计划；
- 审查由开发部门进行的单元和集成测试；
- 进行功能测试；
- 进行系统测试；
- 审查用户手册。

（3）可交付的文档

- 测试记录；
- 测试事故报告；
- 测试总结报告。

6. 运行/维护阶段

（1）测试输入

- 已确认的问题报告；
- 软件生命周期。软件生命周期是一个重复的过程。如果软件被修改了，开发和测试活动都要回归到与修改相对应的生命周期阶段。

（2）测试任务

- 监视验收测试；
- 为确认的问题开发新的测试用例；
- 对测试的有效性进行评估。

（3）可交付的文档

可升级的测试用例库。

5.3　测试用例的设计

1. 测试用例

测试用例(Test Case)是为了高效率地发现软件缺陷而精心设计的少量测试数据。实际测试中,由于无法达到穷举测试,所以要从大量输入数据中精选有代表性或特殊性的数据来作为测试数据。好的测试用例应该能发现尚未发现的软件缺陷。

2. 测试用例的内容

测试用例应包含以下内容:

(1) 测试用例表

测试用例表如表 5-1 所示。对其中一些项目做如下说明:

表 5-1　测试用例表

用例编号		测试模块	
编制人		编制时间	
开发人员		程序版本	
测试人员		测试负责人	
用例级别			
测试目的			
测试内容			
测试环境			
规则指定			
执行操作			
测试结果	步骤	预期结果	实测结果
	1		
	2		
	⋮		
备　　注			

- 测试项目:指明并简单描述本测试用例是用来测试哪些项目、子项目或软件特性的。
- 用例编号:对该测试用例分配惟一的标识号。
- 用例级别:指明该用例的重要程度。测试用例的级别分为 4 级:级别 1(基本)、级别 2(重要)、级别 3(详细)、级别 4(生僻)。

- 执行操作：执行本测试用例所需的每一步操作。
- 预期结果：描述被测项目或被测特性所希望或要求达到的输出或指标。
- 实测结果：列出实际测试时的测试输出值，判断该测试用例是否通过。
- 备注。如需要，则填写“特殊环境需求（硬件、软件、环境）”、“特殊测试步骤要求”、“相关测试用例”等信息。

（2）测试用例清单

测试用例清单如表 5-2 所示。

表 5-2 测试用例清单

项目编号	测试项目	子项目编号	测试子项目	测试用例编号	测试结论	结论
1		1		1		
⋮		⋮		⋮		
总数		—			—	—

5.4 测试总结报告

测试总结报告主要包括测试结果统计表、测试问题表和问题统计表、测试进度表、测试总结表等。

1. 测试结果统计表

测试结果统计表主要是对测试项目进行统计，统计计划测试项和实际测试项的数量，以及测试项通过多少、失败多少等。测试结果统计表如表 5-3 所示。

表 5-3 测试结果统计表

项　目	计划测试项	实际测试项	【Y】项	【P】项	【N】项	【N/A】项	备注
数量							
百分比							

其中，【Y】表示测试结果全部通过，【P】表示测试结果部分通过，【N】表示测试结果绝大多数没通过，【N/A】表示无法测试或测试用例不适合。

另外，根据表 5-3，可以按照下列两个公式分别计算测试完成率和覆盖率，作为测试总结报告的重要数据指标。

测试完成率＝实际测试项数量/计划测试项数量×100%

测试覆盖率＝【Y】项的数量/计划测试项数量×100%

2. 测试问题表和问题统计表

测试问题表如表 5-4 所示，问题统计表如表 5-5 所示。

表 5-4　测试问题表

问题号	
问题描述	
问题级别	
问题分析与策略	
避免措施	
备注	

在表 5-4 中，问题号是测试过程所发现的软件缺陷的惟一标号，问题描述是对问题的简要介绍，问题级别在表 5-5 中有具体分类，问题分析与策略是对问题的影响程度和应对的策略进行描述，避免措施是提出问题的预防措施。

表 5-5　问题统计表

问题程度	严重问题	一般问题	微小问题	其他统计项	问题合计
数量					
百分比					—

从表 5-5 得出，问题级别基本可分为严重问题、一般问题和微小问题。根据测试结果的具体情况，级别的划分可以有所更改。例如，若发现极其严重的软件缺陷，可以在严重问题级别的基础上，加入特殊严重问题级别。

3. 测试进度表

测试进度表如表 5-6 所示，用来描述关于测试时间、测试进度的问题。根据表 5-6，可以对测试计划中的时间安排和实际的执行时间状况进行比较，从而得到测试的整体进度情况。

表 5-6　测试进度表

测试项目	计划起始时间	计划结束时间	实际起始时间	实际结束时间	进度描述

4. 测试总结表

测试总结表包括测试工作的人员参与情况和测试环境的搭建模式，并且对软件产品的质量状况做出评价，对测试工作进行总结。测试总结表如表 5-7 所示。

表 5-7 测试总结表

项目编号		项目名称	
项目开发经理		项目测试经理	
测试人员			
测试环境（软件、硬件）			
软件总体描述：			
测试工作总结：			

小　　结

精心设计的测试计划是软件测试成功与否的关键步骤，在软件测试过程中要因情况变化而随时更改测试计划。

完善的测试文档记录了整个测试活动过程，能够为测试工作提供有力的文档支持，对于各个测试阶段都有着非常明显的指导作用和评价作用。测试文档主要分为测试计划类和测试分析报告类。

习　　题

1. 简述测试计划的定义。
2. 概括测试文档的含义。
3. 简述测试计划的制订原则。
4. 简述测试文档的内容。
5. 简述软件生命周期各阶段的测试任务与可交付的文档。
6. 举例说明测试用例的设计方法。
7. 选择一个小型应用系统，为其做出系统测试的计划书、设计测试用例并写出测试总结报告。

第 6 章

软件自动化测试

本章概述

软件测试是一项艰苦的工作，工作量大，需要投入大量的时间和精力，而完全的手工测试已经满足不了软件开发的需求。软件自动化测试应运而生。软件自动化测试就是通过自动化测试工具或其他手段，按照测试工程师的预定计划进行自动地测试，目的是减轻手工测试的工作量，从而达到提高软件质量的目的。本章具体介绍了手工测试和自动化测试的区别，重点介绍了常用的自动化测试工具。

6.1 软件自动化测试概述

软件自动化测试是相对于手工测试而存在的，主要是通过所开发的软件测试工具、脚本(Script)等来实现，具有良好的可操作性、可重复性和高效率等特点。测试自动化是软件测试中提高测试效率、覆盖率和可靠性的重要测试手段，也可以说，测试自动化是软件测试不可分割的一部分。

迭代式的开发过程是目前最流行的软件开发过程。在迭代式开发中强调在较短的时间间隔中产生多个可执行、可测试的软件版本，这就意味着测试人员也必须为每次迭代产生的软件系统进行测试。测试工作的周期被缩短了，测试的频率增加了。在这种情况下，手工测试已经远远满足不了软件开发的需求。当第一个可测试的版本产生后，测试人员开始对这个版本的系统进行测试，很快第二个版本在第一个版本的技术基础上产生了，测试人员需要在第二次测试时重复上次的测试工作，还要对新增加的功能进行测试，每经过一个迭代，测试工作量就会逐步累加。随着软件开发过程的进展，测试工作越来越繁重，如果使用手工测试的方法，将很难保证测试工作的进度和质量。在这种情况下，使用良好的自动化测试工具势在必行。测试人员可以根据测试需求完成测试过程中所需的行为，使用自动化测试工具自动生成测试脚本。在后续的测

试过程中，只需要对测试脚本进行简单的修改，就完全可以重复使用，而不必手工重复已经测试过的功能部分。

6.2 自动化测试的设计与开发

6.2.1 自动化测试的产生及定义

测试软件是一项艰苦的工作，工作量很大，需要投入大量的时间和精力。据统计，测试工作会占用整个软件开发时间的40%，对于一些可靠性要求很高的软件，测试时间甚至占到总开发时间的60%。但是，我们知道，软件测试具有一定的重复性。通常，如果要测试某项特性，可能需要不止一次的测试，除了要检查前面的测试中发现的软件故障和缺陷是否得到了修复和改进，同时还要检查在修复过程中是否又引入了新的故障或者缺陷。而此后软件又不断地升级，还要进行很多次的重复测试，在这种情形下，软件自动化测试技术开始逐步产生，并不断地发展起来。

自动化测试就是希望通过自动化测试工具或其他手段，按照测试工程师的预定计划进行自动地测试，目的是减轻手工测试的工作量，从而达到提高软件质量的目的。

软件自动化测试是软件测试的重要组成部分，它能完成许多手工测试无法实现的或者难以实现的测试，甚至可以提供要比手工测试更好、更快的测试执行方式，可以省去许多繁杂的工作，节省大量的测试时间。实施正确、合理的自动化测试，能够快速、完整地对软件进行测试，从而提高软件的质量，进而提高对整个软件开发工作的质量，并节约软件开发经费，缩短软件产品发布的周期，带来显著的生产效果和经济效益。

6.2.2 手工测试与自动化测试

虽然自动化测试是软件测试不可分割的一部分，但自动化测试并不能完全取代手工测试，二者各有优缺点。如表6-1所示显示了手工测试与自动化测试的比较结果。这个测试案例中包括1750个测试用例和700多个错误。

表6-1 自动化测试和手工测试比较

测试步骤	手工测试	自动化测试	通过使用工具改善测试的百分比(%)
测试计划的开发	32	40	−25
测试用例的开发	262	117	55
测试执行	466	23	95
测试结果分析	117	58	50
错误状态/更正检测	117	23	80
产生报告	96	16	83
时间总和	1090	277	75

通过表 6-1 可以看出，自动化测试与手工测试在很多方面都有很大的不同，在执行测试和产生测试报告方面显得尤为突出。

通常手工测试的目的着重于发现新的软件故障，而自动化测试的目的则着重于发现旧的软件故障。

1. 手工测试的局限性

测试人员在进行手工测试时，具有创造性，可以举一反三，从一个测试用例想到另外一些测试用例，特别是可以考虑到测试用例不能覆盖的一些特殊的或边界的情况。同时，对于那些复杂的逻辑判断、界面是否友好，手工测试具有明显的优势。但是手工测试在某些测试方面，可能还存在着一定的局限性。包括：

(1) 通过手工测试无法做到覆盖所有代码路径。

(2) 简单的功能性测试用例在每一轮测试中都不能少，而且具有一定的机械性、重复性。其工作量往往较大，无法体现手工测试的优越性。

(3) 许多与时序、死锁、资源冲突、多线程等有关的错误通过手工测试很难捕捉到。

(4) 在系统负载、性能测试时，需要模拟大量数据或大量并发用户等各种应用场合时，也很难通过手工测试来进行。

(5) 在进行系统可靠性测试时，需要模拟系统运行十年、几十年，以验证系统能否稳定运行，这也是手工测试无法模拟的。

(6) 如果有大量的测试用例，需要在短时间内完成，手工测试几乎不可能做到。

(7) 测试可以发现错误，并不能表明程序的正确性。因为不论黑盒测试、白盒测试的方式都不能实现穷举测试。对一些关键程序，则需要考虑利用数学归纳法或谓词演算等进行证明。

2. 自动化测试的好处

由于手工测试的局限性，软件测试借助测试工具极为必要，并向软件测试全面自动化方向发展，将测试工具和软件测试自动化结合起来，解决手工测试的局限性。好的自动化测试可以达到比手工测试更有效、更经济的效果。自动化测试的优点如下：

(1) 对程序的新版本运行回归测试。对于产品型的软件，每发布一个新的版本，其中大部分功能和界面都和上一个版本相似或完全相同，这部分功能可用已有的测试，即回归测试。新版本的测试特别适合于自动化测试，从而达到可以重新测试每个功能的目的。这是最主要的任务，特别是经过了频繁的修改后，一系列回归测试的开销是最小的。假设已经有一个测试在程序的一个老版本上运行过，那么在几分钟之内就可以选择并执行自动化测试。

(2) 可以运行更多更频繁的测试。自动化测试的最大好处就在于，可以在较少的时间内运行更多的测试。例如，产品的发布周期是 3 个月，在测试期间要求每天或每 2 天就要发布一个版本供测试人员测试，一个系统的功能点有几千个或几万个，如果使用人工测

试来完成这么多烦琐的工作,将需要花费大量的时间,难以提高测试效率。

(3) 可以进行一些手工测试难以完成或不可能完成的测试。有些非功能性方面的测试,如压力测试、并发测试、大数据量测试、崩溃性测试,手工测试是不可能实现的。例如,对于200个用户的联机系统,用手工进行并发操作的测试几乎是不可能的,但用自动化测试工具就可以模拟来自200个用户的输入。客户端用户通过定义可以自动回放的测试,随时都可以运行用户脚本,技术人员即使不了解整个内容复杂的商业应用也可以胜任。另外,在测试中应用测试工具,可以发现正常测试中很难发现的缺陷。例如,Numega的DevPartner工具就可以发现软件中的内存方面的问题。

(4) 充分地利用资源。将频繁的测试任务自动化,如需要重复输入数据的测试。这样可以将测试人员解脱出来,提高准确性和测试人员的积极性,把更多的精力投入到测试用例的设计当中。由于使用了自动化测试,手工测试就会减少,相对来说测试人员就可以把更多的精力投入到手工测试过程中,有助于更好地完成手工测试。另外,测试人员还可以利用夜间或周末及其空闲的时候执行自动化测试。

(5) 测试具有一致性和可重复性。由于每次自动化测试运行的脚本是相同的,所以每次执行的测试具有一致性,很容易发现被测软件是否有修改之处。这在手工测试中是很难做到的。另外,好的自动化测试机制还可以确保测试标准与开发标准的一致性。

(6) 测试具有复用性。但对于一些要重复使用的自动化测试要确保其可靠性。

(7) 缩短软件发布的时间。一旦一系列自动化测试准备工作完成,就可以重复地执行一系列的测试,因此能够缩短测试时间。

(8) 增强软件的可靠性。

3. 自动化测试的局限性

上面列举的是自动化测试的优点。既然自动化测试有这么多的优点,那是不是手工测试就可以省略了呢?答案是否定的。自动化测试并不能完全取代手工测试。因为与任何事物一样,自动化测试也有它的不完美之处。自动化测试的缺点如下:

(1) 并非所有的测试都可以用自动化测试来实现。最简单的一个例子就是软件的使用性能测试。软件的使用性能测试必须要求能够判断所测试的软件是否符合人们对一般软件使用时形成的习惯及共识,而这一点是自动化测试无法做得到的,因为它没有人类的感官,又比如操作系统或网络的设置测试、兼容性测试,这些采用手工测试比较合适,因为许多因素都是随机的,必须由测试人员通过判断来执行。

(2) 新的软件缺陷越多,自动化测试失败的几率就越大。发现更多的新的软件缺陷应该是手工测试的主要目的。测试专家James Bach认为85%的软件缺陷靠手工测试来发现,而自动化测试只能发现15%的软件缺陷。其实自动化测试能够很好地发现原来就有的软件缺陷。

(3) 技术问题、组织问题、脚本维护问题。自动化测试实施起来并不简单。首先,商

用测试执行工具是较庞大且复杂的产品，要求具有一定的技术知识，才能很好地利用工具，这对于厂商或分销商培训直接使用工具的用户，特别是自动化测试用户来说十分重要。除工具本身的技术问题外，用户也要了解被测试软件的技术问题。如果软件在设计和实现时没有考虑可测试性，则测试时无论是采用自动化测试还是手工测试难度都非常大。如果使用工具测试这样的软件，无疑更增加测试的难度。其次，还必须有管理支持及组织艺术。最后，还要考虑组织是否能够重视，是否能成立这样的测试团队，是否有这样的技术水平，当测试脚本的维护工作量很大时，是否值得维护等问题。

(4) 测试工具与其他软件的互操作性。测试工具与其他软件的互操作性也是一个严重的问题，技术环境变化如此之快，使得厂商很难跟上。许多工具看似理想，但在某些环境中却并非如此。

4. 自动化测试的适用情况

综上所述，自动化测试与手工测试各有优缺点，应该是互补、并存的。既然是这样，究竟哪些测试应该编成自动测试程序呢？根据自动化测试的特点来看，以下的测试应该优先考虑编写成自动测试：

(1) 回归测试。回归测试是用于保证软件某一时期的质量水平、验证某些缺陷已被修复及修复并没有影响到软件其他部分。由于回归测试是软件每次有新版本出来的时候都必须执行的，也就是在软件的生命周期中会被反复执行的测试，因此这类测试很适合编成自动化测试。

(2) 设计大量不同数据输入的功能测试。如各种各样的边界值测试，过程冗长需要大量时间去完成的网页大量链接的测试等。

(3) 用手工测试完成难度较大的测试。如性能测试、负载测试、强度测试等。例如，对于一个网站软件，要测试 3000 名用户在某一时间内同时在该网站搜索时，服务器运行是否正常及速度是否仍然可以接受。这类测试是手工测试难以完成的，而采用相关的测试软件则很容易做到。

6.2.3　测试工具的运用及作用

1. 测试工具的运用

(1) 测试用例的生成

用编程语言或更专业的脚本语言(如 Perl、PHP、Java 等)编写出的小程序来产生大量的测试输入，包括输入数据与操作指令。或同时也按一定的逻辑规律产生标准输出，输入与输出的文件名字按规定进行配对，使控制自动化测试及结果的核对易于操作。

对于测试用例的命名，如果在项目的文档设计中做了统一规划，则软件产品的需求与功能的命名方法将会有利于后继开发过程的中间产品的命名分类。这样，对文档管理和配置管理都会带来很多方便，使得软件产品开发有条理，符合思维逻辑。

(2) 测试的执行与控制

单元测试可能多用于单机运行,但对于系统测试或回归测试,则可能需要在多机网络环境下进行。利用自动化测试,无论是单机运行还是多机运行,主要的功能和作用是节约大量时间与人力、物力,提高效率并降低成本。

对程序的反复修改、重新汇编和重新测试,如果采用手工方法所花费的时间相当可观,利用软件测试工具就可节约大量时间了。

对于系统测试或者回归测试这类涉及大量测试个案运行的情况,节约测试时间策略方法除了利用自动化工具外,就是如何充分利用一切硬件资源,将大量的测试个案分配到各台机器上同时去运行(并行方式),并将大量的系统测试运行安排在夜间和周末进行。

(3) 测试结果与标准输出的对比

在设计测试用例时,必须考虑到如何能够易于对测试结果实现标准输出。输出数据量的多少及数据格式对比较的速度将会有直接的影响。

另一方面,考虑输出数据与测试用例的目标逻辑对应性及易读性。通常需要写一些特殊的程序来执行测试结果与标准输出的对比工作,因为有的部分输出内容是不能直接对比的。例如,对运行的日期时间的记录、对运行的路径的记录以及测试数据的版本等。

(4) 不吻合测试结果的分析处理

用于对测试结果与标准输出进行对比的自动化工具,往往也同时对不吻合的测试结果能够进行分析、分类、记录和报告工作。这里:

① 分析是找出不吻合的地方并指出错误的可能原因。

② 分类包括各种统计上的分项。例如,对应的源程序的位置、错误的严重级别(提示、警告、非实效性错误、时效性错误或其他分类方法)。

③ 记录是分类的存档。

④ 报告是主动地对测试的运行者及测试用例责任人通报出错的信息。最直接的通报方法是由自动化测试软件发出电子邮件给测试运行者和测试用例负责人。

(5) 测试状态的统计和报表的产生

这是运用自动化测试所应完成的任务,目的是提高过程管理的质量,同时节约用于产生统计数据的时间。通常自动化测试工具均有此项功能。

(6) 自动化测试与开发中产品每日构件的配合

自动化测试是整个开发过程中的一个有机部分。自动化测试要依靠配置管理来提供良好的运行环境,同时与开发中的软件构建紧密配合。通常,在开发的软件产品达到一定程度时,就要开始进行每日测试。这种方法能使软件的开发状态得到频繁更新,及早发现设计和集成中的故障与缺陷。

(7) 采用自动比较技术

测试验证是检验软件是否产生了正确输出的过程,是通过在测试的实际输出与预期

输出(例如,当软件正确执行时的输出)之间完成一次或多次比较来实现的。进行测试自动化工作时,自动比较就成为一个必须的环节。有计划地进行比较会比随意的比较有更高的效率和发现问题的能力。

自动比较的内容可以是多方面的,包括基于磁盘输出的比较,如对数据文件的比较;基于界面输出的比较,如对显示位图的比较;基于多媒体输出的比较,如对声音的比较;还包括其他输出内容的比较。

比较器可以检测两组数据是否相同,功能较齐全的比较器还可以标识有差异的内容。但比较器并不能告诉测试者测试是否通过或失败,需要测试者自行判断。

比较也可以是简单的比较,仅匹配实际输出与预期输出是否完全相同,这是自动化比较的基础。智能比较是允许用已知的差异来比较实际输出和预期输出。例如,要求比较包含日期信息的输出报表的内容。如果使用简单比较,显然是不行的,因为每次生成报表的日期信息肯定是不同的。这时就需要智能比较,忽略日期的差别,比较其他内容,甚至还可以忽略日期的具体内容,但比较日期的格式,则要求日期按特定格式输出。智能比较需要使用到较为复杂的比较手段,包括正则表达式的搜索技术、屏蔽的搜索技术等。

2. 测试工具的作用

自动化测试工具可以帮助开发人员和用户了解以下重要的信息。

(1) 确定系统最优的硬件配置:大量的硬件如何进行配置才能提供最好的性能。

(2) 检查系统的可靠性:整个系统在怎样的负载下能可靠运行,运行的时间有多久,系统的性能会如何变化。

(3) 检查系统硬件和软件的升级情况:软件和硬件对系统性能的影响有多大。

(4) 评估新产品:新的软件产品应当采用哪些新的硬件和软件才能支持运行。

6.2.4 自动化测试产生的问题

对测试工具能够发挥的作用大家都已经很清楚了,但是在软件测试自动化的实施过程中还是会遇到许多问题,以下是一些比较常见的问题。

1. 不现实的期望

没有建立一个正确的软件测试自动化观念,认为测试自动化可以代替手工测试,认为测试自动化可以发现大量新缺陷。实际上,第一次运行的测试最有可能发现缺陷,如果再次运行相同的测试,则发现缺陷的概率就小得多。对回归测试而言,再次运行相同的测试只是确保修改是正确的,并不能发现新的问题。大多数情况下,人们对软件测试自动化存在过于乐观的态度、过高的期望,期望通过这种测试自动化的方案能解决目前遇到的所有问题。但事实上,如果期望不现实,无论测试工具如何,都满足不了期望。

2. 缺乏测试的实践经验

软件测试自动化不仅是简单地使用测试工具,还需要有良好的测试流程,测试用例配

合脚本的编写,这就要求测试人员要很好地掌握测试和编程技术。如果缺乏测试的实践经验,测试组织较差,测试发现缺陷的能力较差,那么首先要做的是改进测试的有效性,而不是改进测试效率。只有手工测试积累到一定程度,才能做好自动化测试。

3. 测试工具本身的问题影响测试质量

自动测试工具本身也是软件产品,如果它的质量得不到保证,将直接影响到测试结果的正确性。不同的测试工具面向不同的测试目的,具有各自的特点和适用范围,因此,一定要根据公司的现实情况来引入正确的测试工具。一般来说,通过自动测试工具测试的用例是不需要再进行手工测试的。将自动测试与手工测试有效地结合,并在最终的测试报告中体现自动测试的结果是比较好的方向。

4. 存在安全性的错觉

如果软件测试工具没有发现被测软件的缺陷,并不能说明软件中不存在问题,可能测试工具本身不够全面或者测试的预期结果设置错误。

5. 自动测试的维护

当软件修改后,通常也需要修改部分测试,这样必然导致对自动化测试的修改。在进行自动化测试的设计和实现时,需要注意这个问题,防止自动化测试带来的好处被高维护成本所淹没。

6.3 常用自动化测试工具简介

随着人们对测试工作的重视以及测试工作的不断深入,越来越多的公司开始使用自动化测试工具。如果能够正确地选择和使用自动化测试工具,就会提高测试的效率和测试质量,降低测试成本。由于一些商用的自动化测试工具十分昂贵,因此在选择自动化测试工具时,要把各种因素考虑进去,只有这样才能做出正确的选择。

6.3.1 自动化测试工具的分类

根据测试方法的不同,自动化测试工具可以分为白盒测试工具、黑盒测试工具和测试管理工具。这些工具主要是 Mercury Interactive(MI)、Segue、IBM/Rational、Compuware 和 Empirix 等公司的产品,而 MI 公司的产品占了主流。这些工具和软件开发过程中相关活动的关系如图 6-1 所示。

1. 白盒测试工具

白盒测试工具一般是针对代码进行测试,测试中发现的缺陷可以定位到代码级。根据测试工具工作原理的不同,白盒测试工具又可以分为静态测试工具和动态测试工具。

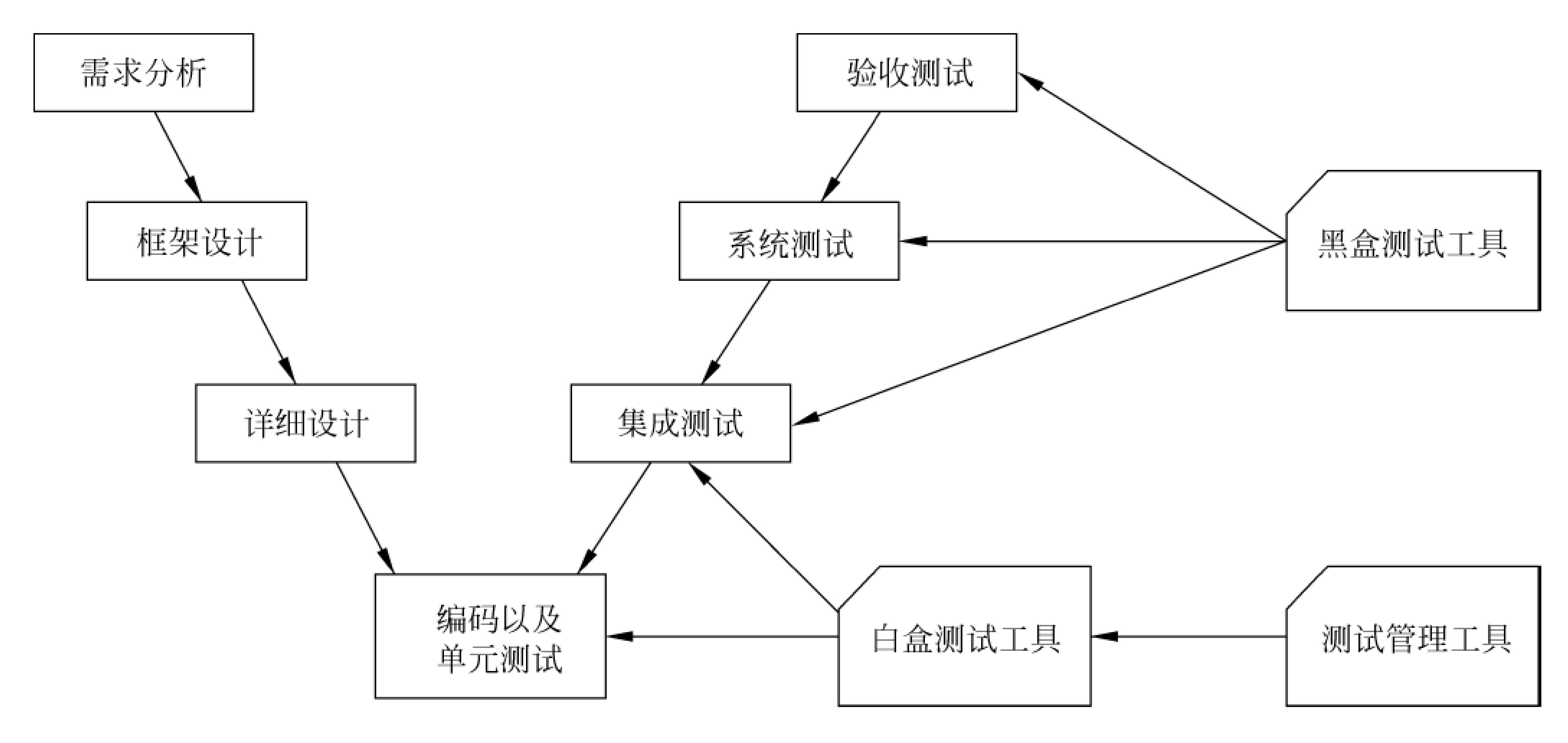


图 6-1　测试工具与开发过程关系图

(1) 静态测试工具

静态测试就是在不执行程序的情况下分析软件的特性。因此，静态测试工具一般是对代码进行语法扫描，找出不符合编码规范的地方，根据某种质量模型评价代码的质量，生成系统的调用关系图等。静态测试工具直接对代码进行分析，不需要运行代码，也不需要对代码编译链接、生成可执行文件。

静态测试工具的代表有 Telelogic 公司的 Logiscope 软件、PR 公司的 PRQA 软件。

(2) 动态测试工具

动态测试直接执行被测程序以提供测试活动。因此，动态测试工具需要实际运行被测系统，并设置断点，向代码生成的可执行文件中插入一些监测代码，掌握断点这一时刻程序运行数据。

动态测试工具的代表有 Compuware 公司的 DevPartner 软件、Rational 公司的 Purify 软件。

常见的白盒测试工具，如表 6-2、表 6-3 所示。

表 6-2　Parasoft 白盒测试工具集

工　具　名	支持语言环境	简　　介
Jtest	Java	代码分析和动态类、组件测试
Jcontract	Java	实时性能监控以及分析优化
C++ Test	C,C++	代码分析和动态测试
CodeWizard	C,C++	代码静态分析
Insure++	C,C++	实时性能监控以及分析优化
.test	.Net	代码分析和动态测试

表 6-3 Compuware 白盒测试工具集

工具名	支持语言环境	简介
BoundsChecker	C++,Delphi	API 和 OLE 错误检查、指针和泄露错误检查、内存错误检查
TrueTime	C++,Java,Visual Basic	代码运行效率检查、组件性能的分析
FailSafe	Visual Basic	自动错误处理和恢复系统
Jcheck	MS Visual J++	图形化的线程和事件分析工具
TureCoverage	C++,Java,Visual Basic	函数调用次数、所占比率统计以及稳定性跟踪
SmartCheck	Visual Basic	函数调用次数、所占比率统计以及稳定性跟踪
CodeReview	Visual Basic	自动源代码分析工具

2. 黑盒测试工具

黑盒测试工具适用于系统功能测试和性能测试,包括功能测试工具、负载测试工具、性能测试工具等。黑盒测试工具的一般原理是利用脚本的录制(Record)/回放(Playback),模拟用户的操作,然后将被测系统的输出记录下来同预先给定的标准结果比较。黑盒测试工具可以大大减轻黑盒测试的工作量,在迭代开发的过程中,能够很好地进行回归测试。

黑盒测试工具的代表有 Rational 公司的 TeamTest、Compuware 公司的 QACenter。

常见的黑盒功能测试工具,如表 6-4 所示。

表 6-4 常见的黑盒功能测试工具

工具名	公司名	官方站点
WinRunner	Mercury Interactive	http://www.merc-inc.com
Astra Quicktest	Mercury Interactive	http://www.merc-inc.com
LoadRunner	Mercury Interactive	http://www.merc-inc.com
Robot	IBM/Rational	http://www-306.ibm.com/software/rational/
TeamTest	IBM/Rational	http://www-306.ibm.com/software/rational/
QARun	Compuware	http://compuware.com
QALoad	Compuware	http://compuware.com
SilkTest	Segue Software	http://www.segue.com
SilkPerformer	Segue Software	http://www.segue.com
e-Test	Empirix	http://www.empirix.com
e-Load	Empirix	http://www.empirix.com
WAS	MS	http://www.microsoft.com
WebLoad	Radview	http://www.radview.com
OpenSTA	OpenSTA	http://www.opensta.com

3. 测试管理工具

测试管理工具用于对测试进行管理。一般而言，测试管理工具负责对测试计划、测试用例、测试的实施进行管理。另外，测试管理工具还能实现对产品缺陷的跟踪管理、产品特性管理等。

测试管理工具的代表有 Rational 公司的 TeamManager、Compuware 公司的 TrackRecord、Mercury Interactive 公司的 TestDirector 等软件。

除此之外，还有专用于性能测试的工具包括有：Radview 公司的 WebLoad；Microsoft 公司的 WebStress 等工具；针对数据库测试的 TestBytes；对应用性能进行优化的 EcoScope 等工具。

6.3.2　常见自动化测试工具介绍

1. Rational Robot

Rational Robot 可以对在各种独立开发环境(IDE)中开发的应用程序，创建、修改并执行功能测试、分布式功能测试、回归测试以及整合测试，记录并回放能识别业务应用程序对象的测试脚本，可以快速、有效地跟踪、报告与质量保证测试相关的所有信息，并将这些信息绘制成图表。Robot 的回归测试与 Purify 结合使用可以完成可靠性测试，与 PureCoverage 结合使用可以完成代码覆盖计算，与 Rational Quantify 结合使用可以完成应用程序性能的测试。

Rational Robot 是一个面向对象的软件测试工具，主要针对 Web、ERP 和 C/S 进行功能自动化测试。可以降低在功能测试上的人力和物力的投入成本和风险，测试包括可见的和不可见的对象。

Rational Robot 可以开发运用三种测试脚本：用于功能测试的 GUI 脚本、用于性能测试的 VU 以及 VB 脚本。

Rational Robot 具有以下功能和作用。

(1) 执行完整的功能测试。记录和回放遍历应用程序的脚本以及测试在查证点处的对象状态。

(2) 执行完整的性能测试。通过 Rational Robot 与 Rational Test Manager 的协作可以记录和回放脚本，这些脚本帮助断定多客户系统在不同负载情况下是否能够按照用户定义的标准运行。

(3) 在 SQA Basic、VB、VU 多种环境下创建并编辑脚本。Rational Robot 编辑器提供有色代码命令，并在集成脚本开发阶段提供键盘帮助。

(4) 测试微软 IDE 环境下 VB、HTML、Java、Oracle Forms、PowerBuilder、Delphi 等编程工具开发的应用程序以及用户界面上看不见的那些对象。

(5) 脚本回放阶段收集应用程序诊断信息。Rational Robot 与 Rational Purify

Quantify PureCoverage 集成，可以通过诊断工具回放脚本，并在日志中查看结果。

由于 Robot 使用面向对象的记录技术，它记录对象内部的名称而非屏幕坐标。因此，当对象改变位置或者窗口文本发生变化时，Robot 仍然可以找到对象并进行回放。

（6）同 Rational 其他组件或产品集成使用 Robot。

- Rational Administrator：用于集中管理 Rational 项目。
- Rational Test Manager 和 Comparators：用于回顾和分析测试结果。
- Rational SiteCheck：用于管理因特网和企业网网页站点。
- Rational TestFactory：测试应用程序。
- Rational ClearQuest：管理缺陷。
- Rational TestManager：测试性能。
- Rational RequisitePro：做需求管理。

2. WinRunner

Mercury Interactive 公司的 WinRunner 是一种企业级的用于检验应用程序是否如期运行的功能性测试工具。通过自动捕获，检测和模拟用户交互的操作，WinRunner 能够辨认缺陷并且确保那些跨越多个应用程序和数据库的业务流程在初次发布时就能避免出现故障，并且保持长期可靠的运行。

WinRunner 的特点在于：与传统的手工测试相比，它能快速、批量地完成功能点测试；能针对相同测试脚本，执行相同的动作，从而消除人工测试所带来的理解上的误差；此外，它还能重复执行相同动作，测试工作中最枯燥的部分可交由机器完成；它支持程序风格的测试脚本，一个高素质的测试工程师能借助它完成流程极为复杂的测试，通过使用通配符、宏、条件语句、循环语句等，还能较好地完成测试脚本的重用；它针对于大多数编程语言和 Windows 技术，提供了较好的集成、支持环境，这对基于 Windows 平台的应用程序实施功能测试而言带来了极大的便利。

WinRunner 具有如下的主要功能：

（1）轻松创建测试。用 WinRunner 创建一个测试，只需使用鼠标和键盘，完成一个标准的业务操作流程，WinRunner 自动记录操作并生成所需的脚本代码，可以直接修改测试脚本以满足各种复杂测试的需求。

（2）插入检查点。在记录一个测试的过程中，可以插入检查点，检查在某个时刻/状态下，应用程序是否运行正常。在插入检查点后，WinRunner 会收集一套数据指标，在测试运行时对其一一验证。WinRunner 可以提供几种不同类型的检查点，包括文本、GUI、位图和数据库。

（3）检验数据。除了创建并运行测试，WinRunner 还能验证数据库的数值，从而确保业务交易的准确性。例如，在创建测试时，可以设定哪些数据库表和记录需要检测；在测试运行时，测试程序就会自动核对数据库内的实际数值和预期的数值。WinRunner 自动

显示检测结果，在有更新/删除/插入的记录上突出显示以引起注意。

(4) 增强测试。为了彻底全面地测试一个应用程序，需要使用不同类型的数据来测试。WinRunner 的数据驱动向导(Data Driver Wizard)可以让你简单地点击几下鼠标，就可以把一个业务流程测试转化为数据驱动测试，从而反映多个用户各自独特且真实的行为。

(5) 运行测试。创建好测试脚本，并插入检查点和必要的添加功能后，就可以开始运行测试。运行测试时，WinRunner 会自动操作应用程序，就像一个真实的用户根据业务流程执行着每一步的操作。测试运行过程中，如有网络消息窗口出现或其他意外事件出现，WinRunner 也会根据预先的设定排除这些干扰。

(6) 分析结果。测试运行结束后，WinRunner 通过交互式的报告工具来提供详尽的、易读的报告。报告中会列出测试中发现的错误内容、位置、检查点和其他重要事件，帮助测试人员对测试结果进行分析。这些测试结果还可以通过 Mercury Interactive 的测试管理工具 TestDirector 来查阅。

(7) 维护测试。随着时间的推移，开发人员会对应用程序做进一步的修改，并需要增加另外的测试。使用 WinRunner，不必对程序的每一次改动都重新创建测试。WinRunner 可以创建在整个应用程序生命周期内都可以重复使用的测试，从而大大地节省时间和资源。

3. LoadRunner

Mercury Interactive 的 LoadRunner 是一种适用于企业级系统、各种体系架构的自动负载测试工具，通过模拟实际用户的操作行为和实行实时性能监测，帮助更快地查找和发现问题，预测系统行为并优化系统性能。通过使用 LoadRunner，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。此外，LoadRunner 能支持广泛的协议和技术，为一些特殊环境提供特殊的解决方案。业界认为 LoadRunner 的功能与 QALoad 相比不相上下。

LoadRunner 的主要功能如下。

(1) 轻松创建虚拟用户。LoadRunner 可以记录下客户端的操作，并以脚本的方式保存，然后建立多个虚拟用户，在一台或几台主机上模拟上百或上千虚拟用户同时操作的情景，同时记录下各种数据，并根据测试结果分析系统瓶颈，输出各种定制压力测试报告。

(2) 使用 Virtual User Generator，能简便地创立起系统负载。该引擎能生成虚拟用户，以虚拟用户的方式模拟真实用户的业务操作行为。利用虚拟用户，在不同的操作系统的机器上同时运行上万个测试，从而反映出系统真正的负载能力。

(3) 创建真实的负载。LoadRunner 能建立持续且循环的负载，限定负载又能管理和驱动负载测试方案，而且可以利用日程计划服务来定义用户在什么时候访问系统以产生负载，使测试过程高度自动化。

(4) 定位性能问题。LoadRunner 内含集成的实时监测器,在负载测试过程的任何时候,可以观察到应用系统的运行性能,实时显示交易性能数据和其他系统组件的实时性能。

(5) 分析结果以精确定位问题所在。测试完毕,LoadRunner 收集、汇总所有的测试数据,提供高级的分析和报告工具,以便迅速查找到问题并追溯原由。

(6) 此外,LoadRunner 完全支持基于 Java 平台应用服务器 Enterprise Java Beans 的负载测试,支持无限应用协议 WAP 和 I-mode,支持 Media Stream 应用,可以记录和重放任何流行的多媒体数据流格式来诊断系统的性能问题,查找原由、分析数据的质量。

4. Parasoft C++ Test

Parasoft C++ Test 是 Parasoft 公司开发的专门针对 C/C++ 的源程序代码进行自动化单元测试的工具,可以自动测试任何 C/C++ 函数、类,自动生成测试用例、测试驱动函数或桩函数,在自动化的环境下完成单元测试,其单元级的测试覆盖率可以达到 100%。Parasoft C++ Test 能够自动测试代码构造(白盒测试)、测试代码的功能性(黑盒测试)和维护代码的完整性(回归测试)。

(1) 白盒测试。Parasoft C++ Test 的白盒测试针对 C/C++ 被测源程序进行分析,对所有类的成员函数及公共成员函数以及私有函数进行测试。在进行测试时,对指定的文件、类或者函数自动生成测试用例。当输入非法参数时,判断有关的函数是否能够处理。

(2) 黑盒测试。Parasoft C++ Test 的黑盒测试不对被测程序进行分析,只是对类的公共接口函数进行测试。黑盒测试时,Parasoft C++ Test 不自动生成测试用例,而是直接运行在"测试用例编辑器"中当前已有的测试用例,这时需要测试人员进行手工添加。

(3) 回归测试。回归测试是在修改了被测试的程序之后,用原有的测试用例进行重新测试的活动。Parasoft C++ Test 的回归测试使用较为简单,既可针对 VC 工程进行全面的测试,也可只对一个 C/C++ 的原文件进行测试。在软件项目规模较大的时候,通常按照一个个文件分别进行测试,而不是直接针对一个工程进行。

5. QACenter

QACenter 是黑盒测试工具,它可以帮助测试人员创建一个快速、可重用的测试过程。该测试工具能够自动帮助管理测试过程,快速分析和调试程序,能够针对回归测试、强度测试、单元测试、并发测试、集成测试、移植测试容量和负载测试建立测试用例,自动执行测试并产生相应的测试文档。

QACenter 测试工具主要包括以下几个模块:

(1) QARun。QARun 主要用于客户端/服务器系统中对客户端的功能测试。在功能测试中,主要包括对系统的 GUI 进行测试以及对客户端事务逻辑进行测试。QARun

的测试实现方法是通过鼠标移动、键盘点击活动操作被测系统，得到相应的脚本，并对脚本进行编辑和调试。在记录过程中针对被测系统中所包含的功能点进行基线的建立，也就是说在插入检查点的同时建立期望输出值。一般情况下，检查点在 QARun 提示目标系统执行一系列事件之后被执行，检查点可以确定实际结果与期望结果是否相同。

（2）QALoad。QALoad 是强负载下应用的性能测试工具。它主要检测系统负载能力，支持范围广、测试内容多。该工具能够帮助测试人员、开发人员和系统管理人员对于分布式系统的被测程序进行有效的负载测试。负载测试能够模拟大量的用户并发活动，从而发现大用户负载下对 C/S 系统的影响。

（3）Eco Tools。Eco Tools 是可用性管理工具，在性能测试后完成系统的可用性分析。有很多因素影响系统的可用性，用户桌面、网络、服务器、数据库环境以及各种子组件都能链接在一起，任何一个组件都可能造成整个系统对最终用户的不可使用问题。Eco Tools 工具包括的各种 Agents 可以监控服务器资源，特别是 Windows NT、UNIX 系统、Oracle、Sybase、SQL Server 系统和其他的应用软件系统。

（4）Eco Scope。Eco Scope 是性能优化工具。它能解决在大型企业复杂环境下分析与测量应用系统性能的难题。通过提供应用的性能级别及其支撑架构的信息，帮助部门就如何提高应用系统的性能提出多方面的决策方案。Eco Scope 应用综合软件探测技术无干扰地监控网络，能够自动跟踪 LAN/WAN 上的应用流量，采集详细的性能指标，并将这些信息关联到交互界面中，自动识别低性能的应用系统、受影响的服务器与用户性能低下的程度。用户界面也能以一种智能的方式访问大量的 Eco Scope 数据。所以能较快地找到性能问题的根源。

（5）QADirector。QADirector 是测试的组织设计和创建以及管理工具。它提供应用系统管理框架，使开发者和 QA 工作组将所有测试阶段组合在一起，从而最有效地使用现有测试资料、测试方法和应用测试工具。QADirector 使用户能够自动地组织测试资料，建立测试过程，以便对多种情况和条件进行测试。按正确的次序执行多个测试脚本，记录、跟踪、分析和记录测试结果，并与多个并发用户共享测试信息。

6. WebLoad

WebLoad 是 RadView 公司推出的一个性能测试和分析工具，它让 Web 应用程序开发者自动执行压力测试；WebLoad 通过模拟真实用户的操作，生成压力负载来测试 Web Application 的性能。

用户使用 WebLoad 创建的是基于 JavaScript 的测试脚本，称为议程 Agenda，用来模拟客户的行为，通过执行该脚本来衡量 Web 应用程序在真实环境下的性能。如有需要可以在做负载测试的同时，使用服务器监控工具对服务器端的内容进行记录，那样使负载测试更加全面。

WebLoad 的性能测试流程：

(1) 计划一个负载(压力)会话 Load Session,用 WebLoad Console 创建 Load Templates,将一系列压力事件定义到一个压力会话 Load Session 里。

(2) 创建测试议程 Agenda。用 WebLoad Visual AAT 创建测试议程 Agenda。

(3) 创建压力模板 Load Templates。可用 WebLoad Wizard 创建压力模板,可以创建一个预期性能参数的压力测试模板或手工创建压力模板。

(4) 输出测试报告并分析测试结果:实时查看测试结果;创建集成报告;用 WebLoad Reporter 分析测试结果。

7. Web Application Stress (WAS) Tool

微软的 WAS 允许以不同的方式创建测试脚本:可以通过使用浏览器走一遍站点来录制脚本,也可以从服务器的日志文件导入 URL,或者从一个网络内容文件夹选择一个文件。当然,也可以手工地输入 URL 来创建一个新的测试脚本。

WAS 可以使用任何数量的客户端运行测试脚本,全部由一个中央主客户端来控制。在每一个测试开始前,主客户机透明地执行以下任务:

(1) 与其他所有的客户机通信。

(2) 把测试数据分发给所有的客户端。

(3) 在所有客户端同时初始化测试。

(4) 从所有的客户端收集测试结果和报告。

这个特性非常重要,尤其对于要测试一个需要使用很多客户端的服务器群的最大吞吐量时非常有用。除此之外,WAS 是被设计用于模拟 Web 浏览器发送请求到任何采用了 HTTP 1.0 或 1.1 标准的服务器,而不考虑服务器运行的平台。除了它的易用性外,WAS 还有很多其他有用的特性,包括:

(1) 对于需要署名登录的网站,它允许创建用户账号。

(2) 允许为每个用户存储 Cookies 和 Active Server Pages (ASP)的会话信息。

(3) 支持随机的或顺序的数据集,以用在特定的名称-值对。

(4) 支持带宽调节和随机延迟,以更真实地模拟显示情形。

(5) 支持 SSL (Secure Sockets Layer) 协议。

(6) 允许 URL 分组和对每组的点击率的说明。

(7) 提供一个对象模型,可以通过 Microsoft Visual Basic Scripting Edition (VBScript) 处理或者通过定制编程来达到启动、结束和配置测试脚本的效果。

8. TestDirector

TestDirector 是一套测试管理软件。可以使用它来规范科学的测试管理流程,建立起针对项目的测试方案和计划,消除组织机构间、地域间的障碍,让测试人员、开发人员或其他的 IT 人员通过一个中央数据仓库,在不同地方就能交互测试信息。TestDirector 将

测试过程流水化——从测试需求管理，到测试计划，测试日程安排，测试执行再到出错后的错误跟踪——仅在一个基于浏览器的应用中便可完成，而不需要每个客户端都安装一套客户端程序。

（1）需求管理。程序的需求驱动整个测试过程。TestDirector 的 Web 界面简化了这些需求管理过程，以此可以验证应用软件的每一个特性或功能是否正常。通过提供一个比较直观的机制将需求和测试用例、测试结果和报告的错误联系起来，从而确保能达到最高的测试覆盖率。

（2）测试计划的制定。其 Test Plan Manager 指导测试人员如何将应用需求转换为具体的测试计划，组织起明确的任务和责任，并在测试计划期间为测试小组提供关键要点和 Web 界面来协调团队间的沟通。

（3）手工测试与自动测试的结合。多数的测试项目需要手工测试与自动测试相结合，启用一个自动化切换机制，能让测试人员决定哪些重复的手工测试可转变为自动脚本以提高测试速度。TestDirector 还能简化将手工测试切换到自动测试脚本的转换，并可立即启动测试设计过程。

（4）安排和执行测试。一旦测试计划建立后，TestDirector 的测试实验室管理为测试日程制订提供一个基于 Web 的框架。其 Smart Scheduler 能根据测试计划中创立的指标对运行着的测试执行监控，能自动分辨是系统还是应用错误，然后将测试切换到网络的其他机器。使用 Graphic Designer 图表设计，可以很快地将测试分类以满足不同的测试目的，如功能性测试、负载测试、完整性测试等。

（5）缺陷管理。TestDirector 的出错管理直接贯穿作用于测试的全过程，从最初发现问题，到修改错误，再到验证修改结果。利用出错管理，测试人员只需进入一个 URL，就可汇报和更新错误，过滤整理错误列表并作趋势分析。

（6）图形化和报表输出。TestDirector 常规化的图表和报告帮助对数据信息进行分析，还以标准的 HTML 或 Word 形式提供生成和发送正式测试报告。测试分析数据还可简便地输入到标准化的报告工具，如 Excel、ReportSmith、CrystalReports 和其他类型的第三方工具。

6.4　性能测试实例

为了使读者更深入地了解软件测试自动化以及自动化测试工具的使用，本节列举了一个使用 LoadRunner 进行的性能测试实例，希望对初次接触自动化测试的读者能够有所帮助。

该实例是针对某培训中心试题库的性能测试。该试题库是用来对参加培训的人员培训结果的一个考核。试题库在培训中心内部 Web 服务器上，假设开设 50 个账号和密码

可供 50 个考生同时参加考试。要求,每台机器只能由一个用户使用,每个用户只能使用各自不同的账号登录考试系统,做完题目后,要求提交考试结果,若在制定的时间内不提交,则系统强制提交考试结果。

但是,一般测试部门不可能有 50 台机器同时进行测试。所以,可以借 Loadrunner 模拟 IP 地址,修改脚本来协助测试。但是,为了保证测试结果,建议使用所有可用的机器进行复测,因为有时候是不可以完全信赖工具的。

6.4.1 现场测试环境

硬件:50 台 PC 机,Web 服务器。

软件:Loadrunner 8.0,Windows 2000,IE 5.0 和 IE 6.0。

人员:质控小组 2 人,执行现场测试。

项目小组 22 人,提供现场环境。

技术小组 1 人,提供技术支持。

6.4.2 测试要求

50 个用户拥有独立 IP 地址,不同的用户及密码登录,试题完成后各自同时提交。

6.4.3 测试内容

50 个用户以不同的用户名和密码登录试题库。试题完成后,提交考试结果。测试考试结果是否能正常提交以及正确评分。

6.4.4 测试方案

1. 完成 20 台实际的 PC 机进行现场测试

(1) 准备工作,并做计划。第一轮测试执行三遍,设定用户考试内容全部同时提交,第一遍全部使用 IE 5.0,第二遍 10 台使用 IE 5.0,10 台使用 IE 6.0,第三遍全部使用 IE 6.0。

(2) At 9:00,20 个用户同时登录系统。

(3) At 9:05,20 个用户同时全部提交。

(4) 分别记录第一轮测试(三遍)的结果。

(5) 第二轮测试准备工作,设定 15 个用户考试内容同时提交,另外 5 个用户延时 5 分钟提交,全部使用 IE 5.0。

(6) At 9:15,20 个用户同时登录系统。

(7) At 9:20,15 个用户同时提交。

(8) At 9:25,剩余 5 个用户同时提交。

(9) 记录第二轮测试结果。

(10) 第三轮测试准备工作,设定 15 个用户考试内容同时提交,另外 5 个用户延时

5 分钟提交,全部使用 IE 6.0。

(11) At 9：15,20 个用户同时登录系统。

(12) At 9：20,15 个用户同时提交。

(13) At 9：25,剩余 5 个用户同时提交。

(14) 记录第三轮测试结果。

(15) 第四轮测试准备工作,设定 15 个用户考试内容同时提交,另外 5 个用户延时 5 分钟提交,正常提交用户使用 IE 5.0,延时提交用户使用 IE 6.0。

(16) At 9：15,20 个用户同时登录系统。

(17) At 9：20,15 个用户同时提交。

(18) At 9：25,剩余 5 个用户同时提交。

(19) 记录第四轮测试结果。

(20) 第五轮测试准备工作,设定 15 个用户考试内容同时提交,另外 5 个用户延时 5 分钟提交,正常提交用户使用 IE 6.0,延时提交用户使用 IE 5.0。

(21) At 9：15,20 个用户同时登录系统。

(22) At 9：20,15 个用户同时提交。

(23) At 9：25,剩余 5 个用户同时提交。

(24) 记录第五轮测试结果。

(25) 第六轮测试准备工作,设定 15 个用户考试内容同时提交,另外 5 个用户延时 5 分钟提交,正常提交用户其中 10 个使用 IE 5.0,5 个使用 IE 6.0,延时提交用户使用 IE 5.0。

(26) At 9：15,20 个用户同时登录系统。

(27) At 9：20,15 个用户同时提交。

(28) At 9：25,剩余 5 个用户同时提交。

(29) 记录第六轮测试结果。

(30) 第七轮测试准备工作,设定 10 个用户考试内容同时提交,另外 10 个用户分两次分别延时 5 分钟、15 提交。

(31) At 9：35,20 个用户同时登录系统。

(32) At 9：40,10 个用户同时提交。

(33) At 9：45,剩余的其中 5 个用户同时提交。

(34) At 9：55,剩余的 5 个用户同时提交。

(35) 记录第七轮测试结果,参见第二轮测试～第六轮测试过程分别对 IE 5.0 和 IE 6.0 的情况进行测试。

(36) 第八轮测试准备工作,设定其中 10 个用户不提交,由系统强行提交。

(37) At 10：10,20 个用户同时登录系统。

(38) At 10：15,10 个用户同时提交。

(39) 其余用户的内容由系统强行提交。

(40) 记录第八轮测试结果,参见第二轮测试～第六轮测试过程分别对 IE 5.0 和 IE 6.0 的情况进行测试。

(41) 第九轮测试准备工作,设定其中 10 个用户同时提交,5 个用户延时 5 分钟提交,其余用户由系统强行提交。

(42) At 10：25,20 个用户同时登录系统。

(43) At 10：30,10 个用户同时提交。

(44) At 10：35,剩余的其中 5 个用户同时提交。

(45) 剩余 5 个用户系统强制提交。

(46) 记录第九轮测试结果,参见第二轮测试～第六轮测试过程分别对 IE 5.0 和 IE 6.0 的情况进行测试。

2. 模拟 20 个用户(含 10 台 PC 机)进行测试

其中,10 台是 PC 机,另外 10 台机器的 IP 地址是 LoadRunner 模拟出来的。

(1) 在 10 台实际的 PC 机中抽取其中一台虚拟 10 个 IP 地址,包括自身的 IP 地址,该机器上共 11 个 IP 地址,这 11 个 IP 地址只能全部使用 IE 5.0 或者全部使用 IE 6.0。

(2) 其余 9 台实际的 PC 机分别由 9 个人操作,另外一台机器由一位质控部人员操作。

(3) 对于异常情况,延时提交和强制提交全部由实际的机器来模拟。

(4) 其余过程参见测试方案 1。

3. 模拟 20 个用户(含 5 台 PC 机)进行测试

其中,5 台是 PC 机,另外 15 台机器的 IP 地址是用 LoadRunner 模拟出来的。

(1) 在 5 台实际的 PC 机中抽取其中一台虚拟 15 个 IP 地址,包括自身的 IP 地址,该机器上共 16 个 IP 地址,这 16 个 IP 地址只能全部使用 IE 5.0 或者全部使用 IE 6.0。

(2) 其余 4 台实际的 PC 机分别由 4 个人操作,另外一台机器由一位质控部人员操作。

(3) 对于异常情况,延时提交和强制提交全部由实际的机器来模拟。

(4) 其余过程参见测试方案 1。

4. 模拟 35 个用户进行测试

其中,20 台是 PC 机,另外 15 台机器的 IP 地址是用 LoadRunner 模拟出来的。

(1) 在 20 台实际的 PC 机中抽取其中两台分别虚拟 7 个、8 个 IP 地址,这 17 个 IP 地址只能全部使用 IE 5.0 或者全部使用 IE 6.0。

(2) 其余 18 台实际的 PC 机分别由 18 个人操作,另外两台机器由两位质控部人员操作。

(3) 对于异常情况,延时提交和强制提交全部由实际的机器来模拟。

(4) 其余过程参见测试方案 1。

5. 模拟 50 台用户进行测试

其中,20 台是 PC 机,另外 30 台机器的 IP 地址是用分别用两台实际的 PC 机模拟出来的。记录测试结果。

(1) 在 20 台实际的 PC 机中抽取其中两台分别虚拟 15 个 IP 地址,这 32 个 IP 地址只能全部使用 IE 5.0 或者全部使用 IE 6.0。

(2) 其余 18 台实际的 PC 机分别由 18 个人操作,另外两台机器由两位质控部人员操作。

(3) 对于异常情况,延时提交和强制提交全部由实际的机器来模拟。

(4) 其余过程参见测试方案 1。

6. 重复测试方案 5

对测试方案 5 中所述情况重复测试两次。

7. 50 台 PC 机现场测试

为了保证结果的正确性,完成 50 台实际的 PC 机进行现场测试。过程参见测试方案 1。

6.4.5 测试过程

注意:该测试过程针对虚拟 IP 地址情况。

(1) 一台 PC 机上创建 15 个虚拟的 IP 地址。

首先,启动 IP Wizard,过程如下:开始程序→LoadRunner→Tools→IP Wizard。

单击 Add 按钮,添加你计划虚拟的 IP 地址。但是注意不能添加已经被占用的 IP 地址。

(2) 启动 Virtual User Generator,并录制脚本,由于 50 个用户的账号和密码各不相同,所以,要修改脚本,设置参数。

(3) 启动 LoadRunner Controller,先将刚才保存的脚本添加进来。然后单击 Scenario 菜单,激活其中的 Enable IP Spoofer。

(4) 单击屏幕右方的 Generators,添加已经建立的 IP,然后建立连接。

(5) 对连接起来的不同用户(IP 地址)分配不同的脚本,在 Controller 中的 design 中,单击 Load Generators 其中,每个脚本有一个用户执行。

(6) 执行 Scenario。

LoadRunner 内含集成的实时监测器,在负载测试过程中的任何时候,都可以观察到应用系统的运行性能。这些被动监测器将实时显示交易性能数据,如反应时间,和其他系统组件包括应用服务器、Web 服务器、网络设备和数据库等的即时性能。这样,使用者就

可以在测试过程中从客户和服务器的双方面评估这些系统组件的运行性能，从而更快地发现问题。再者，利用 LoadRunner 的 contentCheckTM，可以判断负载下的应用程序功能正常与否。ContentCheck 在 virtual users 运行时，检测应用程序的网络内容，从中确定是否有错误内容传送出去。实时浏览器提供从终端用户角度观察到的程序性能情况。

当测试运行结束后，LoadRunner 收集汇总所有的测试数据，提供高级分析和汇报数据，这样便能迅速查找到性能问题并追溯原因。

小　结

软件测试在整个软件开发过程中占据了将近一半的时间和资源。通过在测试过程中合理地引入软件测试工具，能够缩短软件开发时间，提高测试质量，从而更快、更好地为用户提供需要的软件产品。软件测试自动化具有良好的可操作性、可重复性和高效率等特点，是软件测试中提高测试效率、覆盖率和可靠性等的重要测试手段。软件测试工具能在测试过程中发挥多大的作用，取决于测试过程的管理水平和人员的技术水平。测试过程的管理水平和人员的技术水平都是人的因素，是一个开发组织不断改进，长期积累的结果。

测试工具根据测试方法的不同，可以分为白盒测试工具、黑盒测试工具和测试管理工具等。选择测试工具不仅要遵守一定的程序和步骤，而且要注重测试工具的特性，结合自己的实际应用特点，做出决定。

测试自动化实现到何种程度为好？

(1) 测试自动化的程度再高都不可能取代手工测试，即测试工具不可能取代测试人员；

(2) 一般来讲，测试自动化在整个测试过程中只能占到 30%左右；

(3) 实现、运用自动化的程度还取决于各方面的资源，特别是软件的行业规范性和软件开发的稳定性；

(4) 对于部分白盒测试可以使用测试工具，如对代码性能分析等。

对于某一个实际产品，如何实现测试自动化的计划？

(1) 将测试的基本管理形成自动化，如 bug 管理等；

(2) 利用测试自动化工具来实现一些手工无法进行的测试活动，如：压力测试、并发测试、强度测试等；

(3) 利用测试自动化工具来完成回归测试中的缺陷跟踪测试；

(4) 利用测试自动化工具来记录两个版本的异同，以找出缺陷；

(5) 将整个回归测试都用自动化脚本保存，以完成每次的回归测试；

(6) 而对于白盒测试则可以引入测试工具进行代码分析。

习　题

1. 简述软件测试自动化的意义和作用。
2. 简述自动化测试和手工测试有什么主要区别。
3. 自动化测试工具大致可以分为几类？举例说明几种与之相对应的测试工具。
4. 简述对常用自动化测试工具的认识。
5. 试从网络上免费下载自动化测试工具，并在客户机或服务器上安装和运行。
6. 下载本章所介绍的白盒测试、黑盒测试、负载测试等工具，学习其主要功能及使用方法，尝试进行实例测试。

第 7 章

软件测试管理

本章概述

软件测试是为了尽可能多地发现软件中的缺陷并将其修复，从而提高软件整体质量。软件测试工作不仅要有计划地进行，而且需要科学地组织和管理，这样才能开发出高质量的软件产品。只有对测试活动进行组织策划和有效管理，才能使软件测试在软件质量体系保障中发挥重要作用。

本章从软件质量保证开始，介绍了相关的 ISO 9000 质量体系标准和能力成熟度模型（CMM），阐述了软件测试的组织结构和管理工作。

7.1 软件质量保证与软件测试

7.1.1 软件质量保证

任何软件产品的开发都要围绕质量问题，高质量是研发人员追求的目标。质量是体现在开发过程中，通过科学的管理得到控制。对于一个软件产品，质量保证活动是对整个软件过程的管理，并且要贯穿始终。

软件质量保证就是通过对软件产品有计划地进行检查和审计来验证软件是否合乎标准，找出改进的方法，以达到防止产生软件缺陷的目的。许多软件公司都设有质量保证小组或部门，他们负责审查软件设计和开发人员的活动，客观验证它们是否符合规定的标准和要求。

软件质量保证包括以下几个方面的工作：

（1）参与制定软件质量要求

软件质量保证部门在产品开发的需求分析阶段就开始介入，和软件开发人员一起进行软件需求分析，对软件周期各个阶段制定相关的计划、规范和结束标准，提出可能存在的问题，综合软件各方面特点确定软件要满足的质量要求，形成正式文档，作为各阶段审

查的依据。

(2) 组织正式评审

利用在需求分析阶段和研发人员一起制定的软件质量要求，对软件工程各个阶段的进度、完成的质量情况以及存在的问题进行正式评审，保证每个阶段都遵守已制定的计划、规范和结束标准。如果发现某些流程没有达到质量要求，应按照逐级解决的原则进行解决，并将处理结果通知相关技术人员，把问题的解决过程和结果生成报告，作为以后工作的重要参考文档。

(3) 软件测试管理

软件质量保证工作的一个重要内容就是对测试的管理。软件测试管理的好坏，直接影响到测试的效果。质量保证人员要从整体上监督管理软件测试计划的实施和软件测试策略的应用，依据测试结束标准严格审查，保证整个测试工作高效进行。

(4) 对软件的变更进行控制

软件开发过程中，修改和变更是不可避免的。随着软件问题的出现，研发人员要进行一些有益的修改工作，使软件设计更理想化。但是，一些可能导致新故障的因素甚至新的软件缺陷也会随之而来，这样就对软件质量保证产生了副作用。因此必须严格审查软件开发中的变更请求，认真研究新的变更可能给软件开发带来的影响以及与软件其他部分的冲突，对软件的变更进行控制。

(5) 对软件质量进行度量

软件质量度量是指通过量化软件产品中的每一个质量因素，对各因素进行定量测量，从而得到对软件质量特性的整体评价。软件的质量特性包括功能性、可靠性、易用性、效率、可维护性和可移植性，这些特性确定了需要度量软件质量中的哪些因素。

软件质量度量作为一种函数/功能，输入的是软件数据，输出的是代表质量特性的某一属性值。这样质量保证人员和领导层就能掌握各种指标的量化信息，便于严格控制项目的进程和资源的调配，也有利于在软件开发过程中做出准确的决策。

(6) 对软件质量情况及时记录和报告

软件质量情况的文档化是软件质量保证工作结果的重要体现。

软件质量要求的制定要作记录，软件各阶段的审查情况要形成报告，对软件测试的管理情况也要文档化。这样才能为软件过程的正常运行和不断改进提供有价值的参考资料。

7.1.2　软件质量保证与软件测试的关系

软件测试能够找出软件缺陷，确保软件产品满足需求。但是测试不是质量保证，二者并不等同。测试可以查找错误并进行修改，从而提高软件产品的质量。软件质量保证则是避免错误以求高质量，并且还有其他方面的措施以保证质量问题。可以说软件质量保证与软件测试之间既存在共同点又有不同之处。

从共同点的角度看，软件测试和软件质量保证的目的都是尽力确保软件产品满足需

求，从而开发出高质量的软件产品。两个流程都贯穿在整个软件开发生命周期中。正规的软件测试系统主要包括：制定测试计划、测试设计、实施测试、建立和更新测试文档。而软件质量保证的工作主要为：制定软件质量要求、组织正式审查、软件测试管理、对软件的变更进行控制、对软件质量进行度量、对软件质量情况及时记录和报告。

软件质量保证的职能是向管理层提供正确的可行信息，从而促进和辅助设计流程的改进。软件质量保证的职能还包括监督测试流程，这样测试工作就可以被客观地审查和评估，同时也有助于测试流程的改进。

二者的不同之处在于软件质量保证工作侧重对软件开发流程中的各个过程进行管理与控制，杜绝软件缺陷的产生。而测试则是对已产生的软件缺陷进行修复。

软件质量的一个不可忽视的威胁因素来自软件的修改和变更。所以软件测试是软件质量保证的关键步骤。测试可以发现故障，从而帮助开发者发现问题并纠正问题。测试是任何质量保证过程中必需的但不是所有的部分。对于一个系统测试得越多，就越能确保这一系统的正确性，然而测试通常不能保证整个系统运转的完全正确。因此，测试在保证质量方面的主要职责是找出那些在设计开始时就本应该避免的错误并进行修复。软件质量保证的任务首先是避免错误，要做到这一点，除了测试外还需要其他方面的处理。

7.2 测试的组织管理和测试团队的职责

随着软件开发规模的增大、复杂程度的增加，以寻找软件中的错误为目的的测试工作就显得更加困难。统计表明，开发较大规模的软件，有40%以上的精力是耗费在测试上的，即使富有经验的程序员，也难免在编码中发生错误，何况有些错误在设计甚至分析阶段就已埋下祸根，无论是早期潜伏下来的错误或编码中新引入的错误，若不及时排除，轻者降低软件的可靠性，重者导致整个系统的失败。为了尽可能多地找出程序中的错误，生产出高质量的软件产品，加强对测试工作的组织和管理就显得尤为重要。

7.2.1 软件测试的组织

1. 测试的过程及组织

根据软件测试计划，由一位对整个系统设计熟悉的设计人员编写测试大纲，明确测试的内容和测试通过的准则，设计完整合理的测试用例，以便系统实现后进行全面测试。当软件由开发人员完成并检验后，提交测试组，由测试负责人组织测试，测试一般可以采用下列方式来组织：

（1）编写测试大纲、测试用例

测试人员要仔细阅读有关资料，包括规格说明、设计文档、使用说明书及在设计过程中形成的测试大纲、测试内容及测试的通过准则，全面熟悉系统，编写测试计划，设计测试

用例，做好测试前的准备工作。

（2）将测试过程分阶段

软件测试过程按各测试阶段的先后顺序可分为单元测试、集成测试、确认（有效性）测试、系统测试和验收（用户）测试 5 个阶段。

- 单元测试：测试执行的开始阶段。测试对象是每个单元。测试目的是保证每个模块或组件能正常工作。单元测试主要采用白盒测试方法，检测程序的内部结构。
- 集成测试：也称组装测试。在单元测试基础上，对已测试过的模块进行组装，进行集成测试。测试目的是检验与接口有关的模块之间的问题。集成测试主要采用黑盒测试方法。
- 确认测试：也称有效性测试。在完成集成测试后，验证软件的功能和性能及其他特性是否符合用户要求。测试目的是保证系统能够按照用户预定的要求工作。确认测试通常采用黑盒测试方法。
- 系统测试：在完成确认测试后，为了检验它能否与实际环境（如软硬件平台、数据和人员等）协调工作，还需要进行系统测试。可以说，系统测试之后，软件产品基本满足开发要求。
- 验收测试：测试过程的最后一个阶段。验收测试主要突出用户的作用，同时软件开发人员也应该参与进去。

2. 测试人员组织

人是测试工作中最有价值也是最重要的资源，没有一个合格的负责人、积极的测试小组，测试就不可能实现。为高质高效地完成测试任务，应该组织测试人员进行集体学习，做到如下几点：

- 测试项目的负责人必须做到：把要做的事情理清楚；把要达到的目的说清楚；把做事情的思路和方法理清楚；把合理的资源调配到合适的位置上，让兴趣和能力结合。只有从大的方面先将这些事情理清楚了，才可能使得一个团队具有非常的战斗力。测试项目的负责人应该组织测试人员定期进行培训，让团队的每个人都具备应有的沟通能力、技术能力、自信心、怀疑精神、自我督促能力和洞察力。
- 组织测试人员进行工作总结，在什么地方容易犯错误，犯什么类型的错误，犯错误的原因是什么。那么就需要对各种错误进行统计，以找到问题的根本原因。就问题而讨论问题，问题的实质出在哪里，然后改进。
- 组织测试人员提出意见。如果一个团队要发展，就需要大家一起努力，但是实际做起来很难。要避免一言堂，让大家充分参与到设计中，在其中找到自己的价值，这样每一个人才能关心项目的每一个角落，工作才能更有效率。

3. 软件测试文件组织

软件测试文件描述要执行的软件测试及测试的结果。由于软件测试是一个很复杂的过程,同时也是设计软件开发其他一些阶段的工作,对于保证软件的质量和运行有着重要意义,必须把对它们的要求、过程及测试结果以正式的文件形式写出。测试文件的编写是测试工作规范化的一个组成部分。测试文件不只在测试阶段才考虑,它在软件开发的需求分析阶段就开始着手,因为测试文件与用户有着密切的关系。在软件设计阶段的一些设计方案也应在测试文件中得到反映,以利于设计的检验。测试文件对于测试阶段工作的指导与评价作用更是非常明显的。需要特别指出的是,在已开发的软件投入运行的维护阶段,常常还要进行再测试或回归测试,这时仍然需要用到测试文件。

(1) 测试文件的类型

根据测试文件所起的作用不同,通常把测试文件分成两类,即测试计划和测试分析报告。测试计划详细规定测试的要求,包括测试的目的和内容、方法和步骤,以及测试的准则等。由于要测试的内容可能涉及软件的需求和软件的设计,因此必须及早开始测试计划的编写工作。通常,测试计划的编写从需求分析阶段开始,到软件设计阶段结束时完成。测试分析报告用来对测试结果进行分析说明,经过测试后,证实了软件具有的能力,以及它的缺陷和限制,并给出评价的结论性意见,这些意见既是对软件质量的评价,又是决定该软件能否交付用户使用的依据。由于要反映测试工作的情况,自然要在测试阶段内编写。

(2) 测试文件的使用

测试文件的重要性表现在以下几个方面。

- 验证需求的正确性:测试文件中规定了用以验证软件需求的测试条件,研究这些测试条件对弄清用户需求是十分有益的。
- 检验测试资源:测试计划不仅要用文件的形式把测试过程规定下来,还应说明测试工作必不可少的资源,进而检验这些资源是否可以得到,即它的可用性如何。如果某个测试计划已经编写出来,但所需资源仍未落实,那就必须及早解决。
- 明确任务的风险:有了测试计划,就可以弄清楚测试可以做什么,不能做什么。了解测试任务的风险有助于对潜伏的可能出现的问题事先做好思想上和物质上的准备。
- 生成测试用例:测试用例的好坏决定着测试工作的效率,选择合适的测试用例是做好测试工作的关键。在测试文件编制过程中,按规定的要求精心设计测试用例有重要的意义。
- 评价测试结果:测试文件包括测试用例,即若干测试数据及对应的预期测试结果,完成测试后,将测试结果与预期的结果进行比较,便可对已进行的测试提出评价意见。
- 再测试:测试文件规定和说明的内容对于后面维护阶段由于各种原因的需求进行再测试时,是非常有用的。

- 决定测试的有效性：完成测试后，把测试结果写入文件，这对分析测试的有效性，甚至整个软件的可用性提供了依据。同时还可以证实有关方面的结论。

(3) 测试文件的编制

在软件的需求分析阶段，就开始测试文件的编制工作，各种测试文件的编写应按一定的格式进行。

7.2.2　软件测试的管理

在 7.1 节中介绍了软件测试是软件质量保证的关键步骤。为了真正做好软件测试工作，系统地建立一个软件测试管理体系是非常重要的，只有这样才能确保软件测试在软件质量保证中发挥应有的关键作用。

建立软件测试管理体系有以下几个方面。

- 确定软件测试的每个阶段：制定测试计划、测试设计、实施测试、建立和更新测试文档以及测试管理。
- 确定阶段间的相互关系。制定测试计划、测试设计、实施测试三个阶段是按顺序依次进行并且相互作用，阶段间衔接是规范化的，即每个阶段有开始标志和结束标志。测试管理是对这三个阶段进行监督和管理。建立和更新测试文档则贯穿于整个测试流程。
- 确定进行各阶段测试所需要的标准和策略，掌握其相关文档。
- 确定监督、管理和控制各测试阶段的准则和方法。
- 确保可以获得必要的资源和信息，以支持测试流程的正常进行和监督工作的顺利开展。
- 为了提高测试质量，实行适当改进措施。

软件测试管理的主要内容如下：

(1) 软件产品的监督和测量

对软件产品的质量特性进行监督和测量，主要依据软件需求规格说明书，验证产品是否满足要求；所开发的软件产品是否可以交付。要预先设定质量度量指标并进行测试，只有符合预先设定的指标才可以交付。

(2) 对不符合要求产品的识别和控制

对于软件测试中发现的软件缺陷，要认真记录它们的属性和处理办法，并进行跟踪，直至最终解决。在修复软件缺陷之后，要再次进行验证测试。

(3) 软件过程的监督和测量

从软件测试中可以获取大量关于软件过程及其结果的数据和信息，它们可用于判断这些过程的有效性，为软件过程的正常运行和持续改进提供决策依据。

(4) 产品设计和开发的验证

通过设计测试用例对需求分析、软件设计、程序代码进行验证，确保程序代码与软件

设计说明书一致，软件设计说明书与需求规格说明书一致。对于验证中发现的不合格现象，同样要认真记录和处理，并跟踪解决。解决之后，也要再次进行验证。

7.2.3 测试团队总的职责

组织一支优秀的测试团队是做好软件测试工作的基本保障。良好的组织结构和人员划分会促进测试工作的顺利开展和实施，提高软件测试的效率和质量，从而大大提高软件产品的开发效率和产品质量。

在科学的管理体系下，软件测试团队各个成员要明确自身责任，既要完成本职工作又要相互协调好，为整个测试流程负责。软件测试人员的基本责任应该包括：

(1) 尽早发现软件产品中的所有问题。

(2) 督促软件开发人员及时解决测试中发现的缺陷。

除了上述两个基本责任，软件测试团队的责任还包括：

(1) 帮助项目管理人员制定合理的产品开发计划。

(2) 对软件产品中的问题进行分析和跟踪调查，形成文档，以便让项目管理人员和相关产品开发人员对当前产品的质量情况有全面的了解。

(3) 协助完善软件开发流程，提高产品开发的效率。

7.2.4 软件开发和测试过程的组织结构与职责划分

如图 7-1 所示是软件开发和测试过程中的组织结构。参与整个软件生产流程的人员种类很多，结构图中列举了代表性的开发和测试人员。其中，产品经理和产品开发代表是核心领导。以软件开发经理为首的开发部门和以软件测试经理为首的测试部分既各有分工又需要相互合作，共同开发软件，确保软件质量符合设计标准。

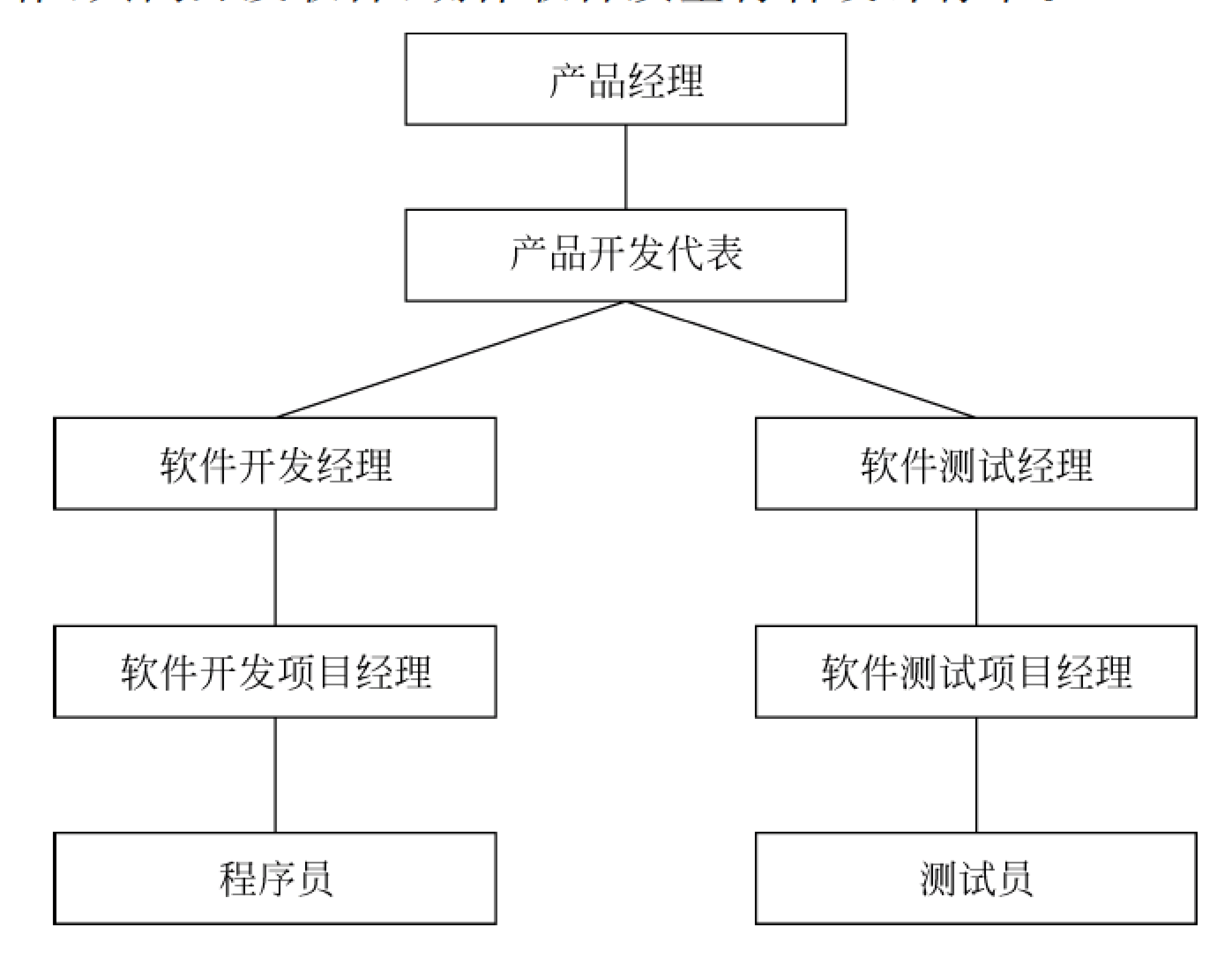


图 7-1 软件开发和测试过程中的组织结构

在需求分析阶段中，软件开发人员的职责如下：

(1) 软件开发项目经理的职责

- 带领项目组分析审核工作任务书；
- 带领项目组与系统工程师进行需求交流并进行分析和文档化；
- 需求跟踪。

(2) 软件开发工程师的职责

- 完成软件需求说明书(SRS)文档；
- 完成需求跟踪；
- 参加 SRS 审查；
- 根据 SRS 评审专家意见，修改 SRS 文档。

(3) 开发代表的职责

- 与项目组一起审查项目任务书；
- 在评审结束后，批准 SRS 文档。

在需求分析阶段中，软件测试人员的职责如下：

(1) 质量保证/软件测试经理的职责

- 监督项目组遵循需求管理流程；
- 参加 SRS 审查；
- 保证相关组参加 SRS 审查。

(2) 软件测试项目经理的职责

- 参与开发人员的软件需求分析，提出可测试性需求；
- 组织人员参与 SRS 的评审工作；
- 组织软件系统测试计划写作；
- 组织软件系统测试方案写作。

(3) 软件测试工程师的职责

- 参与 SRS 评审工作；
- 协助软件测试项目经理完成软件系统测试计划写作；
- 协助软件测试经理完成软件系统测试方案写作。

在软件设计阶段中，软件开发人员的职责如下：

(1) 软件开发项目经理的职责

- 在项目计划中标识设计活动并确保有足够的资源；
- 从项目成员中标识出设计人员，负责设计工作；
- 确保设计人员按照本流程开发相应的设计说明书(HLD 和 LLD)；
- 确保按照审查规程进行设计的审查；
- 批准设计说明书(HLD 和 LLD)；

- 确保更新了需求跟踪矩阵；
- 确保设计文档按照配置管理流程来控制。

(2) 软件开发工程师的职责

- 完成设计文档；
- 完成需求跟踪；
- 参加设计文档审查；
- 根据评审专家意见，修改设计文档。

(3) 相关评审专家的职责

- 针对设计文档，提交评审意见；
- 参加设计文档的评审会议；
- 确认修改后的意见。

在软件设计阶段中，软件测试人员的职责如下：

(1) 质量保证/软件测试经理的职责

- 监督项目组遵循软件设计流程；
- 参加设计审查；
- 保证相关组参加设计审查。

(2) 软件测试项目经理的职责

- 组织所有的测试活动；
- 制定测试策略；
- 确保测试活动有合适的计划；
- 审核并批准单元测试和集成测试的测试计划；
- 确保所有分配需求被跟踪和验证；
- 确保测试策略在签发后基线化，单元测试计划(UTP)、集成测试计划(ITP)、系统测试计划(STP)在审查和批准后基线化。

说明：基线是指一个被正式评审和批准的规格和产品，作为进一步开发的一个基础，并且必须通过正式的变更流程来变更。

(3) 软件测试工程师职责

- 准备测试计划(STP/UTP/ITP)；
- 撰写单元测试(UT)/集成测试(IT)/系统测试(ST)的测试用例；
- 完成需求跟踪。

软件测试执行阶段，软件测试人员的职责如下：

(1) 软件开发项目经理的职责

- 确保缺陷分发给相关软件工程师并及时得到解决；
- 参与需求变更评审。

（2）软件开发工程师的职责

- 修正缺陷；
- 验证相关的缺陷已经被修正。

（3）软件测试项目经理的职责

- 组织所有的测试活动；
- 确保选择适合的测试工具以及测试环境的建立；
- 确保测试活动的计划得到执行和获得资源；
- 确保缺陷分发给相关软件工程师并及时得到解决；
- 审核并批准测试报告；
- 审核并批准测试状态报告。

（4）软件测试工程师的职责

- 搭建测试环境；
- 执行测试用例；
- 将测试中发现的所有缺陷填写在缺陷报告中；
- 回归测试；
- 准备测试报告；
- 测试期间，每周准备测试状态报告。

7.3　ISO 9000 标准

与软件质量管理和质量保证方面相关的是当前国际惯用的 ISO 9000 系列标准。软件测试人员可以以此系列标准规范、约定软件的开发过程，提高软件产品质量。

ISO 9000 系列标准是 ISO 国际标准化组织 TC/176 技术委员会制定的所有国际标准，其中的核心标准是质量保证标准（ISO 9001/2/3）和质量管理标准（ISO 9004）。它们小到螺栓、螺母，大到质量管理和质量保证等所有制造行业设立标准。

具体说，ISO 9001 质量体系是为设计、开发、生产、安装和售后服务提供质量保证模式；ISO 9002 质量体系是为生产和安装提供质量保证模式；ISO 9003 质量体系是为计算机软件的开发、测试、供应、安装和维护提高质量保证模式；ISO 9004 是质量管理和质量体系要素导则。

上述 ISO 9000 系列标准可分为两类：ISO 9001～ISO 9003 作为第一类用于建立客户对生产商质量要求的保证；ISO 9004 作为第二类用于生产商自身建立质量保证体系。ISO 9001、ISO 9002 和 ISO 9003 的作用范围由大到小，ISO 9001 包括从产品开发到售后服务的全程质量要求保证，作用范围最大。ISO 9002 其次，ISO 9003 再次之。

ISO 9000 系列标准的两个主要特点是：

(1) 它的目标在于整个产品流程的控制，从开始设计、生产、产品销售到产品服务，都有相关质量保证规范。如果生产的全过程得到很好的控制并达到约定的质量要求，那么最终的产品质量就得到了保证。

(2) 产品缺陷的提早预防。整个生产过程中有效预防漏洞的出现并且按照系列标准不断进行产品自身完善，做到防患于未然，大大减少甚至杜绝了不合格产品。

ISO 9000 标准中针对软件开发产业的部分是 ISO 9001 和 ISO 9003。ISO 9001 针对设计、开发、生产、安装和服务产品，而 ISO 9003 则针对开发、供应、安装和维护软件产品。ISO 9003 作为软件企业实施 ISO 9001 质量保证模式提供实施指南，它对软件产品从市场调查、需求分析、软件设计、编码、测试等各个开发阶段进行质量保证控制，也对产品发布、销售、成品安装和维护过程进行规范控制，从而保证软件产品的整体质量。

ISO 9003 标准的主要内容：

- 开发详细的质量计划和程序控制配置管理、产品验证、不规范行为（缺陷）和纠正措施（修复）；
- 准备和接收软件开发计划，包括项目定义、产品目标清单、项目进度、产品说明书，如何组织项目的描述，风险和假设的讨论以及控制策略等；
- 使用客户易理解的且测试时易进行合法性检查的用语来表述说明书；
- 计划、开发、编制和实施软件设计审查程序；
- 开发控制软件设计随产品生命周期而发生变化的程序；
- 开发和编制软件测试计划；
- 开发检测软件是否满足客户要求的方法；
- 实施软件验证和接收式测试；
- 维护测试结果的记录；
- 解决软件缺陷的方式；
- 证明产品在发布之前已经就绪；
- 开发控制产品发布过程的程序；
- 明确指出和规定应该收集的质量信息；
- 应用统计技术分析软件开发过程、评估软件产品质量。

7.4 能力成熟度模型（CMM）

CMM（Capability Maturity Model）即软件能力成熟度模型，是软件行业标准模型，用来定义和评价软件公司开发过程的成熟度，为如何提高软件质量提供指导。CMM 是在美国国防部领导下，由软件开发团体和卡耐基－梅隆大学的软件工程学院（SEI）共同开发的，1991 年推出 CMM 1.0 版，1993 年推出 CMM 1.1 版。

CMM 的主要特点是通用性好，它适用于各种规模的软件公司，从大规模的公司集体到个人均可使用。它的 5 个等级为评估软件开发成熟度提供了简单的方法，确定了进入下一个成熟度等级的关键标志。

CMM 将软件过程能力成熟度划分为 5 个等级，如图 7-2 所示。每个等级定义一组过程能力目标，并且表述了能够达到这些过程能力目标的实践活动。依据 CMM 模型，软件开发人员能够更好地按照计划高效率地开发出有质量保证的软件产品。

(1) 等级 1(初始级)

该等级的软件开发过程是随意的，甚至是混乱的。项目能否成功依靠个人的能力和运气，具有偶然性。软件开发过程没有定义，没有通用计划，难以监视和控制。开发的时间和费用无法预知。测试过程和其他过程混杂在一起。

(2) 等级 2(可重复级)

该等级成熟度主要集中在项目级。建立基本的项目管理过程去跟踪成本、进度、功能和质量。可以借鉴以前成功项目的经验，应用到新项目中。软件开发具有一定的组织性，使用了基本软件测试行为，例如测试计划和测试用例。

(3) 等级 3(已定义级)

该等级具备了组织化思想，而不仅仅是针对具体项目。软件开发中的管理活动和工程活动被文档化和标准化，并且形成整个软件组织的标准软件过程。所有项目均在标准软件过程中进行。

(4) 等级 4(已管理级)

在该成熟度等级中，组织过程处于在统计的控制下。软件过程和产品质量有具体的度量标准，软件过程和产品质量得到了定量理解和控制。

(5) 等级 5(优化级)

通过来自过程、新技术和新思想等各方面的定量信息反馈，能够进行持续的过程改进，以期达到质量更佳的等级。

CMM 的分层结构为软件公司开发产品提供了不同等级的可行性目标，达到了某一级设定的等级目标，就达到了这个成熟度等级，自然可以进入下一级。等级 1 是基础，大多数机构的软件过程开发环境还相当不成熟，还都自然处于这个基础上。根据是否真正采用成熟的软件过程来衡量，全世界大多数软件公司的能力成熟度为 1 级，多数为 2 级，少数为 3 级，极少数为 4 级，能力成熟度为 5 级的软件公司更是凤毛麟角。5 个等级如图 7-2 所示，为评测软件公司开发能力成熟度提供了简单的方式。

CMM 和 ISO 9000 系列标准都是用来管理产品的生产过程，监督产品的质量和性能。它们的共同点是注重产品是否符合设计说明书，是否达到用户的预期要求。它们的不同点是，CMM 强调产品生产过程的持续改进，ISO 9000 系列(关于软件开发和测试部分)强调产品达到质量要求的基本准则。当然，在 ISO 9000 系列中也含有关于生产过程

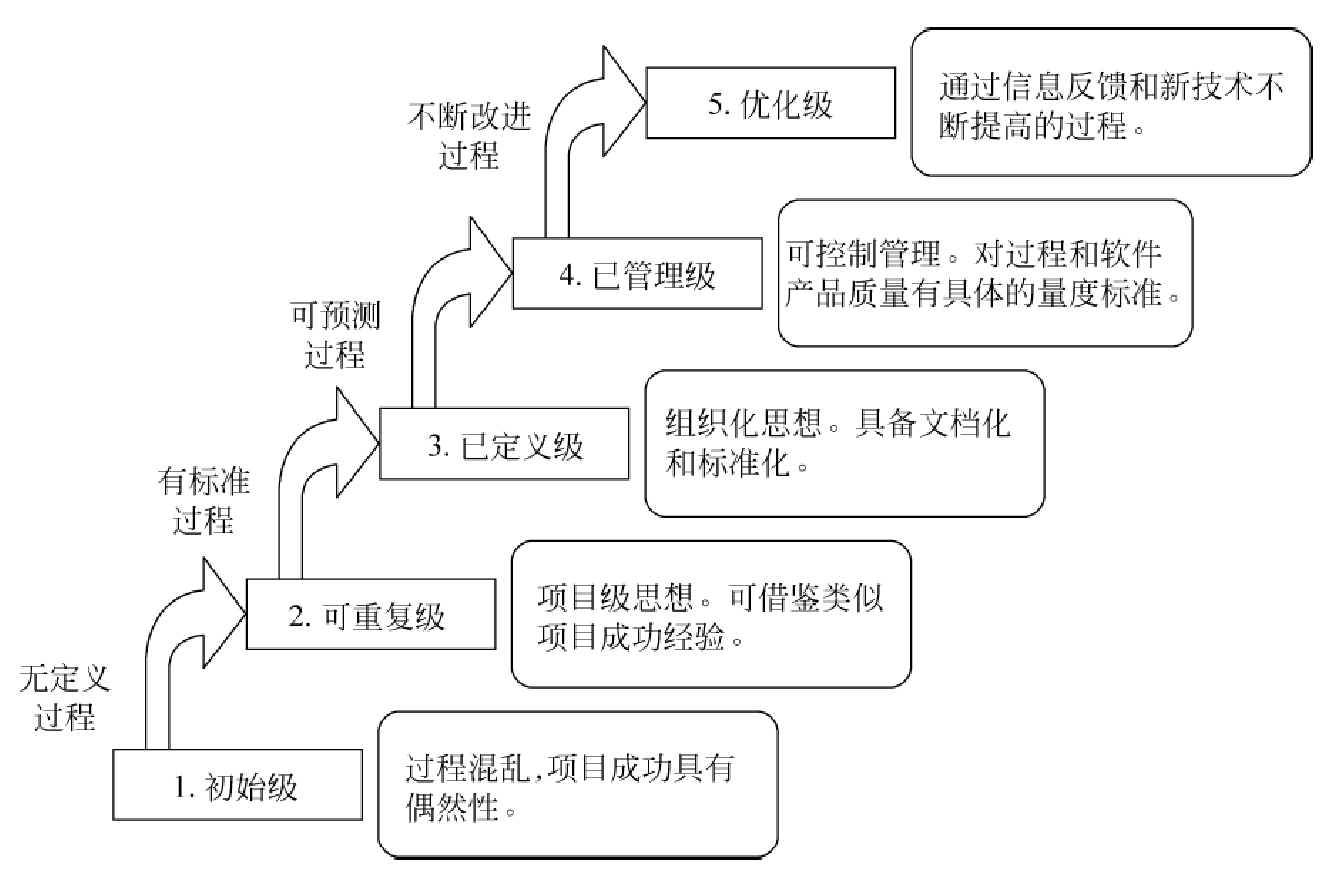


图 7-2　CMM 软件成熟度等级

持续改进的问题的部分。所以,两者是既有联系又有区别。

作为软件公司,不论通过 CMM 评估还是获得 ISO 9000 系列认证,都是为了保证软件开发的质量以及不断改进产品,这样才能满足用户需求,适应市场变化,从而获取竞争优势。

小　　结

软件测试是有组织和有管理的软件质量保证活动,而不是随意的、松散的、杂乱的实施过程。软件质量保证就是通过对软件产品有计划地进行检查和审计来验证软件是否合乎标准,设计改进方案,从而杜绝软件缺陷的产生。软件质量保证部门有责任监督测试流程,保证测试工作的客观性,同时也有助于测试流程的改进。而软件测试是提高软件质量的关键环节。

建立科学的软件测试管理体系是非常重要的,只有这样才能确保软件测试在软件质量保证中发挥应有的重要作用。要想实现良好的测试管理,就要对测试团队进行系统地组织和明确地职责划分。

ISO 9000 系列标准是 ISO 国际标准化组织制定的国际标准,它为所有制造行业设立了质量管理和质量保证等标准。ISO 9003 质量体系是为软件产品的开发和测试提供质量保证模式,可以作为软件测试的质量保证规范。

软件能力成熟度 CMM 作为软件行业的标准模型，可以定义和评价软件公司开发过程的成熟度，为提高软件质量提供指导。

习　　题

1. 质量保证部门与测试部门的职责是否一样？归纳它们的共同点和不同点。
2. ISO 9000 标准与软件测试的关系是什么？
3. 简述 CMM 的具体等级划分。

第 8 章

面向对象的软件测试

本章概述

面向对象程序的结构不再是传统的功能模块结构,它将开发分为面向对象分析、面向对象设计和面向对象编程三个阶段。分析阶段产生整个问题空间的抽象描述,在此基础上,进一步归纳出适用于面向对象编程语言的类和类结构,最后形成代码。针对面向对象软件的开发特点,其测试方法和技术也必然要做相应的改变。本章主要介绍了面向对象软件的特点;面向对象软件测试与传统软件测试的区别;面向对象的测试方法以及类测试用例设计的方法。

8.1 面向对象的软件测试概述

自 20 世纪 80 年代后期以来,面向对象软件开发技术发展迅速,获得了越来越广泛的应用,在面向对象的分析、设计技术以及面向对象的程序设计语言方面,均获得了很丰富的研究成果与工程实际应用。

与之相比较,面向对象软件测试技术的研究还相对薄弱。基于结构的传统集成策略并不完全适于面向对象的程序。这是因为面向对象的程序的执行实际上是执行一个由消息连接起来的方法序列,而这个方法序列往往是由外部事件驱动的。在面向对象语言中,虽然信息隐藏和封装机制使得类具有较好的独立性,有利于提高软件的易测试性和保证软件的质量,但是,信息的隐藏和封装机制与继承机制及动态绑定给软件测试带来了新的课题。尤其是面向对象软件中类与类之间的集成测试和类中各个方法之间的集成测试具有特别重要的意义,与传统语言书写的软件相比,集成测试的方法和策略也应该有所不同。

从目前的研究及测试实际现状来看,较多地集中在类和对象状态的测试方面。面向对象程序设计的继承和动态绑定所带来的多态性对软件测试的影响虽然也有所研究,但

是不仅缺乏针对这一特点的测试方法,而且还有许多问题有待进一步研究。

软件测试中的另一个重要问题是测试的充分性问题,充分性准则对软件测试发现错误的能力具有重要影响。对传统语言的软件测试已经存在多种充分性准则,但对面向对象的软件测试,目前还没有普遍接受的充分性准则。

8.2　面向对象的软件

为了充分说明面向对象软件测试的基本问题和解决策略,首先来回顾一下面向对象及面向对象程序设计的有关基本概念。

(1) 对象

对象是一个可操作的实体,它既包含了特定的数据,也包含了操作这些数据的代码。对象是一个可计算的基本实体,将那些协作解决某类问题的对象组成对象群(集合)。当一个程序被执行时,对象被创建、修改、访问或因为协作的结果而被删除。程序中的对象是一些问题及其解决方法的特定实体的描述。对象具有生命周期。

对象是软件开发期间测试的直接目标。在程序运行时,对象的行为是否符合它的规定说明,该对象与和它相关的对象是否协同工作,这两个方面是面向对象软件测试所关注的焦点。

(2) 类

类是一些具有共性的对象集合。类可看做是创建对象的模板。面向对象程序运行的基本元素是对象,而类则是用来定义对象这一基本元素的。创建对象的过程被称作实例化,创建的结果就是实例。在类中,所有对象的基本概念基本相同,可从两个方面描述:

- 在类的声明中,定义了每个对象能做什么;
- 在类的实现中,定义了类的每个对象如何做它们将做的东西,即描述对象如何表现它的属性。

(3) 接口

接口是行为声明的集合。行为被集中在一起,并通过单个的概念定义相关的动作。接口是由一些规范构成的,规范定义了类的一套完整的公共行为。

从测试的角度,关于接口有下面的结论:

- 接口封装了操作的说明。如果这一接口包含的行为和类的行为不相符,那么这一接口的说明就有问题。
- 接口非孤立,它与其他的接口和类有一定的关系。一个接口可以指定一个行为的参数类型,使得实现该接口的类可被当作一个参数进行传递。

(4) 封装

为有效地使用面向对象的程序方法,首先需要解决程序的结构设计问题。在程序设

计过程中最重要的是抽象，也就是说，从现实世界中抽象出合理的对象结构。在面向对象思想中，抽象决定了对象的属性、内部结构以及处理对象的外部接口。

从程序语言角度来看，在一个对象中代码和(或)数据可以是这个对象私有的，不能被对象外的部分直接访问。因而对象提供了一种高级保护以防止程序被无关部分错误修改或错误地使用了对象的私有部分。当从对象外部试图直接对受保护的内部数据进行修改时，将被程序拒绝，只有通过对象所提供的对外服务函数才能够对其内部数据进行必要的加工，从而保证了数据加工的合法性。从这一意义上讲，把这种代码和数据的联系称为“封装”。换句话说，封装是将对象封闭保护起来，是将内部细节隐蔽起来的能力。

(5) 继承

继承是类的一种联系，允许一个新的类在一个已有的类的基础上进行定义。一个类对另一个类的依赖，使已有的类的说明和实现可被复用。新类主要是子类或继承类(C++中)。被继承的类称作父类或者是基类。

从测试的角度来看，继承包含以下内容：

- 继承提供一种机制，通过这种机制，潜在的错误能够从一个类传递到它的派生类。
- 子类是从父类继承过来的，所以子类也就继承了父类的说明和实现。因此，可以用测试父类的方法对子类进行测试。

(6) 多态

多态提供了将对象看做是一种或多种类型的能力。它定义了用来支持多种不同类型所适应的策略。多态支持灵活的程序设计，同时也易于维护。多态分为包含多态与参数多态。

包含多态是同一个类具有不同表现形式的一种现象，使得参数具有替换的能力。为了响应操作请求，当对象的定义与后续对象的定义相符时，对象就可以被相互替换。换句话说，面向对象程序中的发送者能够在用对象作为参数时根据接口进行实现，而不是实现一个完整的类。

从测试的角度来看，包含多态具有以下功能：

- 包含多态允许系统通过增加类来进行扩展，而非修改已经存在的类。扩展当中可能会出现意外的交互关系。
- 包含多态允许任何操作都可以包括一个或多个类型不确定的参数。这就会增加应该测试的实际参数的种类。

参数多态是能够根据一个或者多个参数来定义一种类型的能力。从测试的角度看，参数多态支持不同类型的继承关系。例如，如果模板仅仅用来初始化单个实例，就不能保证它与其他类很好地协作。

面向对象的程序设计中，针对继承和多态特性的测试是新的测试技术难点。

在面向对象语言中，类是创建对象的关键。类描述了一组对象的公共特征和操作，而

对象则是具体实现的类。

面向对象技术导致程序构件的复用，而复用导致更快的软件开发和高质量的程序。面向对象软件易于维护，因为它的结构是内紧外松，这样当进行修改时，影响面小。此外，面向对象系统易于进行适应性修改及伸缩。归纳起来其优点有：

- 可重用性。从一开始对象的产生就是为了重复利用，完成的对象将在今后的程序开发中被部分或全部地重复利用。
- 可靠性。由于面向对象的应用程序包含了通过测试的标准部分，因此更加可靠。由于大量代码来源于成熟可靠的类库，因而新开发程序的新增代码明显减少，这是程序可靠性提高的一个重要原因。
- 连续性。具有面向对象特点的 C++ 与 C 语言有很大的兼容性，C 程序员可以比较容易地过渡到 C++ 语言开发工作。

1. 信息隐藏对测试的影响

类的重要作用之一是信息隐藏。它对类中所封装的信息的存取进行控制，从而避免类中有关实现细节的信息被错误地使用。而这样的细节性信息正是软件测试所不可忽略的。由于面向对象的软件系统在运行时由一组协调工作的对象组成，对象具有一定的状态，所以对面向对象的程序测试来说，对象的状态是必须考虑的因素，测试应涉及对象的初态、输入参数、输出参数和对象的状态。对象的行为是被动的，它只有在接收有关信息后才被激活来进行所请求的操作，并将结果返回给发送者。在工作过程中对象的状态可能被修改，产生新的状态，面向对象软件测试的基本工作就是创建对象（包括初始化），向对象发送一系列信息，然后检查结果对象的状态，看其是否处于正确的状态。问题是对象的状态往往是隐蔽的，若类中未提供足够的存取函数来表明对象的实现方式和内部状态，则测试者必须增添这样的函数。因此，类的信息隐藏机制给测试带来了困难。

2. 封装和继承对测试的影响

在面向对象的程序中，由于继承的作用，一个函数可能被封装在多个类中，子类中还可以对继承的特征进行重定义。问题是，未重定义的继承特征是否还需要进行测试呢？重定义的特征需要重新测试是显然的，但应该如何测试重定义的特征呢？

Weyuker 的不可分解性公理认为对一个程序进行过充分的测试后，并不表示其中的成分都得到了充分的测试。因此，若一个类得到了充分的测试，当其被子类继承后，继承的方法在子类的环境中的行为特征需要重新测试。

Weyuker 的非复合性公理认为一个测试数据集对于程序中的各程序单元而言都是充分的并不表示它对整个程序是充分的。这一公理表明，若我们对父类中某一方法进行了重定义，仅对该方法自身或其所在的类进行重新测试是不够的，还必须重新测试其他有

关的类(如子类和引用类)。Perry 和 Kaiser 认为封装和继承并未简化测试问题,反而使测试更加复杂。

3. 多态性对测试的影响

传统的软件测试中经常使用静态分析技术对代码进行分析,识别程序执行路径。在面向对象软件中,由于动态绑定和多态性的存在,使得程序可执行的路径急剧增加,由于多态和动态绑定所带来的不确定性,使得传统测试中十分重要的静态分析方法遇到了很大的困难,这种不确定性也给测试覆盖率的满足带来了挑战。

8.3 面向对象测试与传统测试的区别

传统的软件测试策略是从"小型测试"开始,逐步走向"大型测试"。即从单元测试开始,然后逐步进入集成测试,最后是有效性和系统测试。在传统应用中,单元测试集中在最小的可编译程序单位——子程序(如:模块、子例程、进程),一旦这些单元都已经被独立测试过后,就被集成在程序结构中,然后进行一系列的回归测试以发现模块的接口错误和由于新单元的加入而产生的副作用,最后,把系统作为一个整体进行测试以发现需求规格说明中的错误。

面向对象程序的结构不再是传统的功能模块结构,作为一个整体,原有集成测试所要求的逐步将开发的模块搭建在一起进行测试的方法已经不可能。而且,面向对象软件抛弃了传统的开发模式,对每个开发阶段都有不同于以往的要求和结果,已经不可能用功能细化的观点来检测面向对象分析和设计的结果。因此,传统的测试模型对面向对象软件已经不再适用。

面向对象的软件开发模型突破了传统的瀑布模型,将开发分为面向对象分析(OOA)、面向对象设计(OOD)和面向对象编程(OOP)3 个阶段。针对这种开发模型,结合传统的测试步骤的划分,可以把面向对象的软件测试模型分为:面向对象分析的测试(OOA Test)、面向对象设计的测试(OOD Test)、面向对象编程的测试(OOP Test)、面向对象的单元测试(OO Unit Test)、面向对象的集成测试(OO Integrate Test)、面向对象的系统测试(OO System Test)。

面向对象分析的测试和面向对象设计的测试是对分析结果和设计结果的测试,主要针对分析、设计产生的文本进行,是软件开发前期的关键性测试。面向对象编程的测试主要针对编程风格和程序代码实现进行测试,其主要的测试内容在面向对象的单元测试和面向对象的集成测试中体现。

面向对象的单元测试是对程序内部具体单一的功能模块的测试,如果程序是用 C++ 语言实现,主要就是对类成员函数的测试。面向对象的单元测试是进行面向对象的集成测试的基础。面向对象的集成测试主要对系统内部的相互服务进行测试,如成员函数间

的相互作用、类间的消息传递等。面向对象的集成测试不但要基于面向对象的单元测试，更要参见面向对象设计的测试或面向对象设计的测试结果。

面向对象的系统测试是基于面向对象的集成测试的最后阶段的测试，主要以用户需求为测试标准，需要借鉴面向对象分析的测试或面向对象分析的测试的结果。

尽管上述各阶段的测试构成一个相互作用的整体，但其测试的主体、方向和方法各有不同。

1. 面向对象分析的测试(OOA Test)

传统的面向过程分析是一个功能分解的过程，把一个系统看成可以分解的功能的集合。这种传统的功能分解分析法的着眼点在于一个系统需要什么样的信息处理方法和过程，以过程的抽象来对待系统的需要。而面向对象分析(OOA)是"把 E-R 图和语义网络模型，即信息造型中的概念，与面向对象程序设计语言中的重要概念结合在一起而形成的分析方法"，最后通常是得到问题空间的图表的形式描述。

OOA 直接映射问题空间，全面地将问题空间中的现实功能抽象化。将问题空间中的实例抽象为对象，用对象的结构反映问题空间的复杂实例和复杂关系，用属性和操作表示实例的特性和行为。对一个系统而言，与传统分析方法产生的结果相反，行为是相对稳定的，结构是相对不稳定的，这更充分反映了现实的特性。OOA 的结果为后面阶段类的选定和实现、类层次结构的组织和实现提供平台。因此，对 OOA 的测试，应从以下方面考虑：

- 对认定的对象的测试；
- 对认定的结构的测试；
- 对认定的主题的测试；
- 对定义的属性和实例关联的测试；
- 对定义的服务和消息关联的测试。

OOA 中认定的对象是对问题空间中的结构、其他系统、设备、被记忆的事件、系统涉及的人员等实际实例的抽象。对它的测试可以从如下方面考虑：

- 认定的对象是否全面，是否问题空间中所有涉及的实例都反映在认定的抽象对象中。
- 认定的对象是否具有多个属性。只有一个属性的对象通常应看成其他对象的属性，而不是抽象为独立的对象。
- 对认定为同一对象的实例是否有共同的，区别于其他实例的共同属性。
- 对认定为同一对象的实例是否提供或需要相同的服务，如果服务随着不同的实例而变化，认定的对象就需要分解或利用继承性来分类表示。
- 如果系统没有必要始终保持对象代表的实例的信息，提供或者得到关于它的服务，认定的对象也无必要。

- 认定的对象的名称应该尽量准确、适用。

属性是用来描述对象或结构所反映的实例的特性。而实例关联是反映实例集合间的映射关系。对属性和实例关联的测试从如下方面考虑：

- 定义的属性是否对相应的对象和分类结构的每个现实实例都适用；
- 定义的属性在现实世界是否与这种实例关系密切；
- 定义的属性在问题空间是否与这种实例关系密切；
- 定义的属性是否能够不依赖于其他属性被独立理解；
- 定义的属性在分类结构中的位置是否恰当，低层对象的共有属性是否在上层对象属性体现；
- 在问题空间中每个对象的属性是否定义完整；
- 定义的实例关联是否符合现实；
- 在问题空间中实例关联是否定义完整，特别需要注意一对多和多对多的实例关联。

2. 面向对象设计的测试（OOD Test）

通常的结构化的设计方法，用的是面向作业的设计方法，它把系统分解以后，提出一组作业，这些作业以过程实现系统的基础构造，把问题域的分析转化为求解域的设计，分析的结果是设计阶段的输入。

而面向对象设计（OOD）方法采用"造型的观点"，以OOA为基础归纳出类，并建立类结构或进一步构造成类库，实现分析结果对问题空间的抽象。由此可见，OOD是OOA的进一步细化和更高层的抽象。所以，OOD与OOA的界限通常是难以严格区分的。

OOD确定类和类结构不仅是满足当前需求分析的要求，更重要的是通过重新组合或加以适当的补充，能方便实现功能的重用和扩增，以不断适应用户的要求。因此，对OOD的测试，应从如下三方面考虑：

- 对认定的类的测试；
- 对构造的类层次结构的测试；
- 对类库支持的测试。

（1）对认定的类的测试

OOD认定的类可以是OOA中认定的对象，也可以是对象所需要的服务的抽象，对象所具有的属性的抽象。认定的类原则上应该尽量基础性，这样才便于维护和重用。测试认定的类：

- 是否涵盖了OOA中所有认定的对象；
- 是否能体现OOA中定义的属性；
- 是否能实现OOA中定义的服务；
- 是否对应着一个含义明确的数据抽象；

- 是否尽可能少地依赖其他类；
- 类中的方法(C++：类的成员函数)是否单用途。

(2) 对构造的类层次结构的测试

为能充分发挥面向对象的继承共享特性，OOD 的类层次结构，通常基于 OOA 中产生的分类结构的原则来组织，着重体现父类和子类间的一般性和特殊性。在当前的问题空间，对类层次结构的主要要求是能在解空间构造实现全部功能的结构框架。为此，测试如下方面：

- 类层次结构是否涵盖了所有定义的类；
- 是否能体现 OOA 中所定义的实例关联；
- 是否能实现 OOA 中所定义的消息关联；
- 子类是否具有父类没有的新特性；
- 子类间的共同特性是否完全在父类中得以体现。

(3) 对类库支持的测试

对类库的支持虽然也属于类层次结构的组织问题，但其强调的重点是再次软件开发的重用。由于它并不直接影响当前软件的开发和功能实现，因此，将其单独提出来测试，也可作为对高质量类层次结构的评估。拟订测试点如下：

- 一组子类中关于某种含义相同或基本相同的操作，是否有相同的接口(包括名字和参数表)。
- 类中方法(C++：类的成员函数)功能是否较单纯，相应的代码行是否较少(参考文献 5 中建议为不超过 30 行)。
- 类的层次结构是否是深度大，宽度小。

3. 面向对象编程的测试(OOP Test)

典型的面向对象程序具有继承、封装和多态的新特性，这使得传统的测试策略必须有所改变。封装是对数据的隐藏，外界只能通过被提供的操作来访问或修改数据，这样降低了数据被任意修改和读写的可能性，降低了传统程序中对数据非法操作的测试。继承是面向对象程序的重要特点，继承使得代码的重用率提高，同时也使错误传播的概率提高。多态使得面向对象程序对外呈现出强大的处理能力，但同时却使得程序内“同一”函数的行为复杂化，测试时不得不考虑不同类型具体执行的代码和产生的行为。

面向对象程序是把功能的实现分布在类中。能正确实现功能的类，通过消息传递来协同实现设计要求的功能。因此，在面向对象编程(OOP)阶段，忽略类功能实现的细则，将测试的目光集中在类功能的实现和相应的面向对象程序风格上，主要体现为以下两个方面：

- 数据成员是否满足数据封装的要求；
- 类是否实现了要求的功能。

（1）数据成员是否满足数据封装的要求

数据封装是数据和数据有关的操作的集合。检查数据成员是否满足数据封装的要求，基本原则是数据成员是否被外界（数据成员所属的类或子类以外的调用）直接调用。更直观地说，当改编数据成员的结构时，是否影响了类的对外接口，是否会导致相应外界必须改动。值得注意，有时强制的类型转换会破坏数据的封装特性。例如：

```
class Hiden
{private:
int a=1;
char * p= "hiden";}
class Visible
{public:
int b=2;
char * s= "visible";}
…
Hiden pp;
Visible * qq=(Visible * )&pp;
```

在上面的程序段中，pp 的数据成员可以通过 qq 被随意访问。

（2）类是否实现了要求的功能

类所实现的功能，都是通过类的成员函数执行。在测试类的功能实现时，应该首先保证类成员函数的正确性。单独地看待类的成员函数，与面向过程程序中的函数或过程没有本质的区别，几乎所有传统的单元测试中所使用的方法，都可在面向对象的单元测试中使用。

类函数成员的正确行为只是类能够实现要求的功能的基础，类成员函数间的作用和类之间的服务调用是单元测试无法确定的。因此，需要进行面向对象的集成测试。需要着重声明，测试类的功能，不能仅满足于代码能无错运行或被测试类能提供的功能无错，应该以所做的 OOD 结果为依据，检测类提供的功能是否满足设计的要求，是否有缺陷。必要时（如通过 OOD 结果仍不清楚明确的地方）还应该参照 OOA 的结果，以之为最终标准。

4. 面向对象的单元测试（OO Unit Test）

传统的单元测试是针对程序的函数、过程或完成某一定功能的程序块。沿用单元测试的概念，实际测试类成员函数。一些传统的测试方法在面向对象的单元测试中都可以使用。如等价类划分法、因果图法、边值分析法、逻辑覆盖法、路径分析法、程序插装法等。单元测试一般建议由程序员完成。

用于单元级测试进行的测试分析（提出相应的测试要求）和测试用例（选择适当的输

入,达到测试要求),规模和难度等均远小于后面将介绍的对整个系统的测试分析和测试用例,而且强调对语句应该有 100%的执行代码覆盖率。在设计测试用例选择输入数据时,可以基于以下两个假设:

- 如果函数(程序)对某一类输入中的一个数据正确执行,对同类中的其他输入也能正确执行。
- 如果函数(程序)对某一复杂度的输入正确执行,对更高复杂度的输入也能正确执行。例如需要选择字符串作为输入时,基于本假设,就无须计较字符串的长度。除非字符串的长度是要求固定的,如 IP 地址字符串。

在面向对象程序中,类成员函数通常都很小,功能单一,函数间调用频繁,容易出现一些不宜发现的错误。例如:

① “if (-1==write (fid, buffer, amount)) error_out();”该语句没有全面检查 write()的返回值,断然假设只有数据被完全写入和没有写入两种情况。当测试也忽略了数据部分写入的情况,就给程序遗留了隐患。

② 按程序的设计,使用函数 strrchr()查找最后的匹配字符,但在程序中误写成了函数 strchr(),使程序功能实现时查找的是第一个匹配字符。

③ 程序中将“if(strncmp(str1,str2,strlen(str1)))”误写成“if (strncmp(str1,str2,strlen(str2)))”。如果测试用例中使用的数据 str1 和 str2 长度一样,就无法检测出。

因此,在做测试分析和设计测试用例时,应该注意面向对象程序的这个特点,仔细地进行测试分析和设计测试用例,尤其是针对以函数返回值作为条件判断选择,字符串操作等情况。

面向对象编程的特性使得对成员函数的测试,又不完全等同于传统的函数或过程测试。尤其是继承特性和多态特性,使子类继承或过载的父类成员函数出现了传统测试中未遇见的问题。Brian Marick 给出了两方面的考虑:

(1) 继承的成员函数是否都不需要测试

对父类中已经测试过的成员函数,两种情况需要在子类中重新测试:

- 继承的成员函数在子类中做了改动;
- 成员函数调用了改动过的成员函数的部分。

例如,假设父类 Bass 有两个成员函数:Inherited()和 Redefined(),子类 Derived 只对 Redefined()做了改动。Derived∷Redefined()显然需要重新测试。对于 Derived∷Inherited(),如果它有调用 Redefined()的语句(如:x=x/Redefined()),就需要重新测试,反之,无此必要。

(2) 对父类的测试是否能照搬到子类

援用上面的假设,Base∷Redefined()和 Derived∷Redefined()已经是不同的成员函数,它们有不同的服务说明和执行。对此,照理应该对 Derived∷Redefined()重新测试分

析，设计测试用例。但由于面向对象的继承使得两个函数相似，故只需在 Base∷Redefined()的测试要求和测试用例上添加对 Derived∷Redfined()新的测试要求和增补相应的测试用例。

例如：Base∷Redefined()含有如下语句

```
If (value<0) message ("less");
else if (value==0) message ("equal");
else message ("more");
```

Derived∷Redfined()中定义为

```
If (value<0) message ("less");
else if (value==0) message ("It is equal");
else
{message ("more");
if (value==88)message("luck"); }
```

在原有的测试上，对 Derived∷Redfined()的测试只需做如下改动：将 value==0 的测试结果期望改动；增加 value==88 的测试。

多态有几种不同的形式，如参数多态，包含多态，重载多态。包含多态和重载多态在面向对象语言中通常体现在子类与父类的继承关系，对这两种多态的测试参见上述对父类成员函数继承和重载的论述。包含多态虽然使成员函数的参数可有多种类型，但通常只是增加了测试的繁杂。对具有包含多态的成员函数测试时，只需要在原有的测试分析和基础上扩大测试用例中输入数据的类型的考虑。

5. 面向对象的集成测试（OO Integrate Test）

传统的集成测试，是由底向上通过集成完成的功能模块进行测试，一般可以在部分程序编译完成的情况下进行。而对于面向对象程序，相互调用的功能是散布在程序的不同类中，类通过消息相互作用申请和提供服务。类的行为与它的状态密切相关，状态不仅仅是体现在类数据成员的值，也许还包括其他类中的状态信息。由此可见，类相互依赖极其紧密，根本无法在编译不完全的程序上对类进行测试。所以，面向对象的集成测试通常需要在整个程序编译完成后进行。此外，面向对象程序具有动态特性，程序的控制流往往无法确定，因此也只能对整个编译后的程序做基于黑盒子的集成测试。

面向对象的集成测试能够检测出相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误。基于单元测试对成员函数行为正确性的保证，集成测试只关注于系统的结构和内部的相互作用。面向对象的集成测试可以分成两步进行：先进行静态测试，再进行动态测试。

静态测试主要针对程序的结构进行，检测程序结构是否符合设计要求。现在流行的

一些测试软件都能提供一种称为"可逆性工程"的功能，即通过原程序得到类关系图和函数功能调用关系图，例如 International Software Automation 公司的 Panorama-2 for Windows 95、Rational 公司的 Rose C++ Analyzer 等，将"可逆性工程"得到的结果与 OOD 的结果相比较，检测程序结构和实现上是否有缺陷。换句话说，通过这种方法检测 OOP 是否达到了设计要求。

动态测试设计测试用例时，通常需要上述的功能调用结构图、类关系图或者实体关系图为参考，确定不需要被重复测试的部分，从而优化测试用例，减少测试工作量，使得进行的测试能够达到一定覆盖标准。测试所要达到的覆盖标准可以是：达到类所有的服务要求或服务提供的一定覆盖率；依据类间传递的消息，达到对所有执行线程的一定覆盖率；达到类的所有状态的一定覆盖率等。同时也可以考虑使用现有的一些测试工具来得到程序代码执行的覆盖率。

具体设计测试用例，可参考下列步骤：

(1) 先选定检测的类，参考 OOD 分析结果，明确类的状态和相应的行为，类或成员函数间传递的消息，输入或输出的界定等。

(2) 确定覆盖标准。

(3) 利用结构关系图确定待测类的所有关联。

(4) 根据程序中类的对象构造测试用例，确认使用什么输入激发类的状态、使用类的服务和期望产生什么行为等。

值得注意，设计测试用例时，不但要设计确认类功能满足的输入，还应该有意识地设计一些被禁止的例子，确认类是否有不合法的行为产生，如发送与类状态不相适应的消息，要求不相适应的服务等。根据具体情况，动态的集成测试，有时也可以通过系统测试完成。

6. 面向对象的系统测试(OO System Test)

通过单元测试和集成测试，仅能保证软件开发的功能得以实现。但不能确认在实际运行时，它是否满足用户的需要，是否大量存在实际使用条件下会被诱发产生错误的隐患。为此，对完成开发的软件必须经过规范的系统测试。换个角度说，开发完成的软件仅仅是实际投入使用系统的一个组成部分，需要测试它与系统其他部分配套运行的表现，以保证在系统各部分协调工作的环境下也能正常工作。

系统测试应该尽量搭建与用户实际使用环境相同的测试平台，应该保证被测系统的完整性，对临时没有的系统设备部件，也应有相应的模拟手段。系统测试时，应该参考 OOA 分析的结果，对应描述的对象、属性和各种服务，检测软件是否能够完全"再现"问题空间。系统测试不仅是检测软件的整体行为表现，从另一个侧面看，也是对软件开发设计的再确认。

这里说的系统测试是对测试步骤的抽象描述。它体现的具体测试内容包括：

（1）功能测试

测试是否满足开发要求，是否能够提供设计所描述的功能，是否用户的需求都得到满足。功能测试是系统测试最常用和必须的测试，通常还会以正式的软件说明书为测试标准。

（2）强度测试

测试系统的能力最高实际限度，即软件在一些超负荷的情况下，功能实现的情况。如要求软件某一行为的大量重复、输入大量的数据或大数值数据、对数据库大量复杂的查询等。

（3）性能测试

测试软件的运行性能。这种测试常常与强度测试结合进行，需要事先对被测软件提出性能指标，如传输连接的最长时限、传输的错误率、计算的精度、记录的精度、响应的时限和恢复时限等。

（4）安全测试

验证安装在系统内的保护机构确实能够对系统进行保护，使之不受各种非常的干扰。安全测试时需要设计一些测试用例试图突破系统的安全保密措施，检验系统是否有安全保密的漏洞。

（5）恢复测试

采用人工的干扰使软件出错，中断使用，检测系统的恢复能力，特别是通信系统。恢复测试时，应该参考性能测试的相关测试指标。

（6）可用性测试

测试用户是否能够满意使用。具体体现为操作是否方便，用户界面是否友好等。

（7）安装/卸载测试(install/uninstall test)

系统测试需要对被测的软件结合需求分析做仔细的测试分析，建立测试用例。

7. 面向对象的覆盖率

由于传统的结构化度量没有考虑面向对象的一些特性，如多态、继承和封装等，传统的结构化覆盖必须被加强，以满足面向对象特性。上下文覆盖就是一种针对面向对象特性而增强的覆盖。

上下文覆盖可以应用到面向对象领域，处理诸如多态、继承和封装的特性，同时该方法也可以被扩展用于多线程应用。通过使用这些面向对象的上下文覆盖，结合传统的结构化覆盖的方法就可以保证代码的结构被完整地执行，同时提高我们对被测软件质量的信心。

有三个面向对象上下文覆盖的定义，分别是：

- 继承上下文覆盖：该覆盖用于度量在系统中的多态调用被测试的程度。
- 基于状态的上下文覆盖：该覆盖用于改进对带有状态依赖行为的类的测试。
- 已定义用户上下文覆盖：该度量允许上下文覆盖的方法被应用到传统结构化覆盖率无法使用的地方，如多线程应用。

8.4　面向对象的测试方法

1. 继承层次结构中类的测试

继承是实现接口和代码复用的有效机制，也是面向对象语言的一大特色。那么如何对继承层次结构中的类进行有效的测试呢？下面以 Java 语言为例来说明这个问题。

在 Java 中引入的“继承”机制，是面向对象程序设计方法中的一种重要手段，通过继承可以更有效地组织程序结构，明确类间关系，并充分利用已有的类来完成更复杂、深入的开发。根据继承机制的特点，不难推断所有为某个类确定的测试用例集合对该类的子类也是有效的。父类中被测试用例所测试的代码被子类继承，只要父类代码没有被子类“覆盖”，那么就不用重新创建这些测试用例。类的创建是自上而下的，同样在构建测试类时也应采用自上而下的方法。如图 8-1 所示显示了类之间的继承关系。

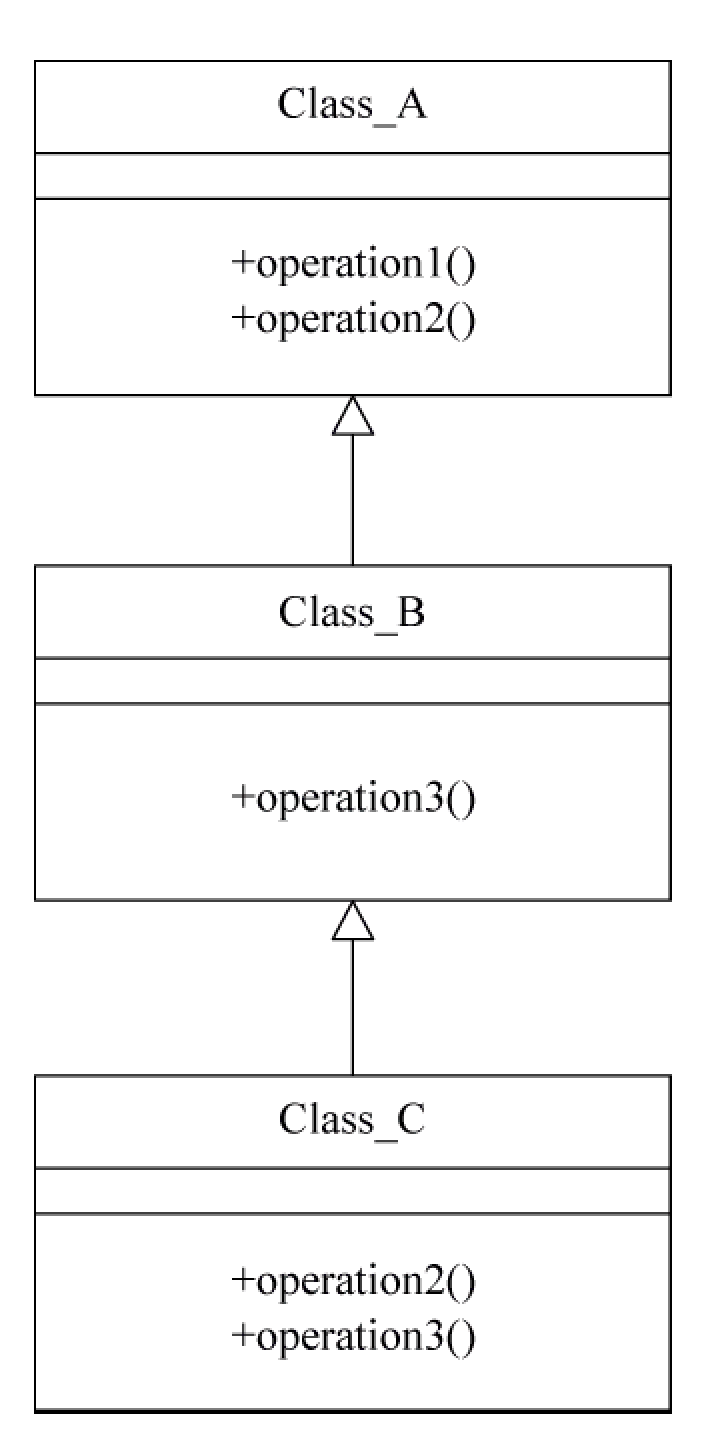


图 8-1　类之间的继承关系

从图 8-1 中可以看出，Class_A 类有两个实例方法 operation1()和 operation2()，Class_B 类继承了 Class_A 类并且实现了新的实例方法 operation3()，Class_C 类继承了 Class_B 类，覆盖了 Class_B 类的实例方法 operation3()和实例方法 operation2()。

根据图 8-1 中这 3 个类之间的区别，就可以确定继承的测试用例中是否需要产生新的子类测试用例，哪些测试用例适用于测试子类，哪些测试用例在测试子类中不必执行，如表 8-1 所示。

表 8-1　子类测试用例增补表

类	继承类	类方法	是否改变	是否增加测试用例
Class_A		operation1()		
		operation2()		
Class_B	Class_A	operation1()	False	False
		operation2()	False	False
		operation3()	True	True
Class_C	Class_B	operation1()	False	False
		operation2()	True	True
		operation3()	True	True

由此，可以得出继承层次结构中类测试的测试用例可以采用如下增补原则：

- 如果子类新增了一个或者多个新的操作，就需要增加相应的测试用例。
- 如果子类定义的同名方法覆盖了父类的方法，就需要增加相应的测试用例。

在具体构建类测试用例时可以采用如图 8-2 所示的结构。对于基类则需要百分之百地进行测试，底层的测试类可以对其父类的测试方法回归。

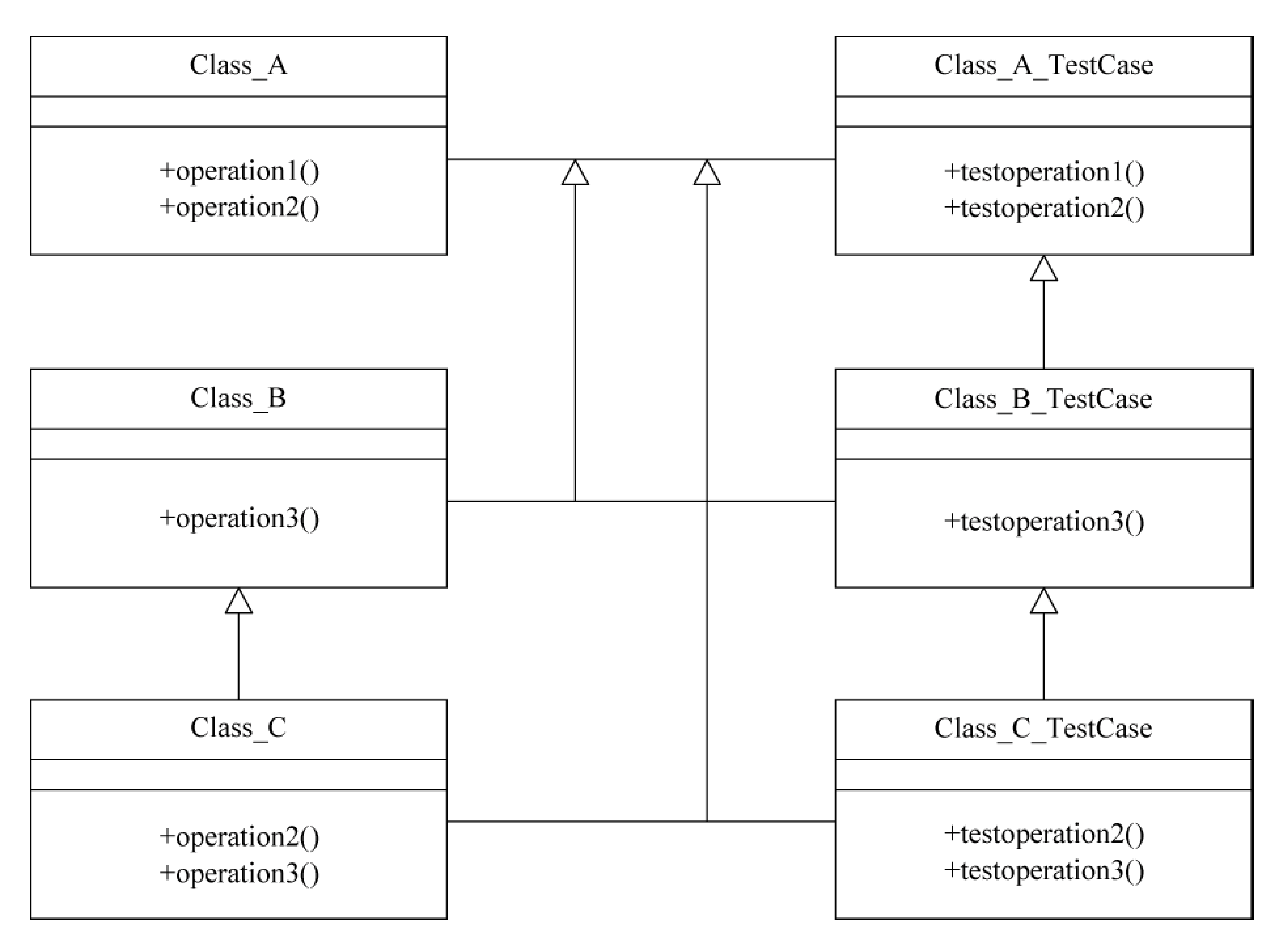


图 8-2　类测试用例的构建

2. 接口类测试

在 Java 语言中，接口仅仅描述方法的特征但并不具体实现，在语法上与类有些相似，可以定义若干个抽象方法和常量，形成一个属性集合，该属性集合通常代表了某一组功能的实现，可以实现与类类似的多重继承功能。在对类进行测试时需要构建可执行的类实例，而接口不存在任何构造方法无法被实现。那么，应该如何对接口进行测试呢？是否应该为接口单独定义一个实现它并且可实例化的测试类呢？由于接口一定会在某个类中实现，所以说这样做是没有必要的，可以使用一个实现接口的类来做测试即可。在对接口类进行测试时，总体来说应该遵循以下原则：

- 如果接口没有被任何类实现就无需进行测试。
- 如果已被别的类实现，那么就针对实现该接口的类进行测试。

如图 8-3 所示是一个接口测试类的例子。从图 8-3 中可以看出，Interface_A 接口定义了公共 a_Method()和 b_Method()方法，该接口被具体类 Class_C 实现，测试类 Class_C_TestCase 同样实现了该接口。那么，Class_C 和 Class_C_TestCase 都必须实现 a_Method()和 b_Method()方法，目的就是为了强制测试人员必须测试所有的接口方法，同时关注被测试类是如何具体实现接口中的方法的。

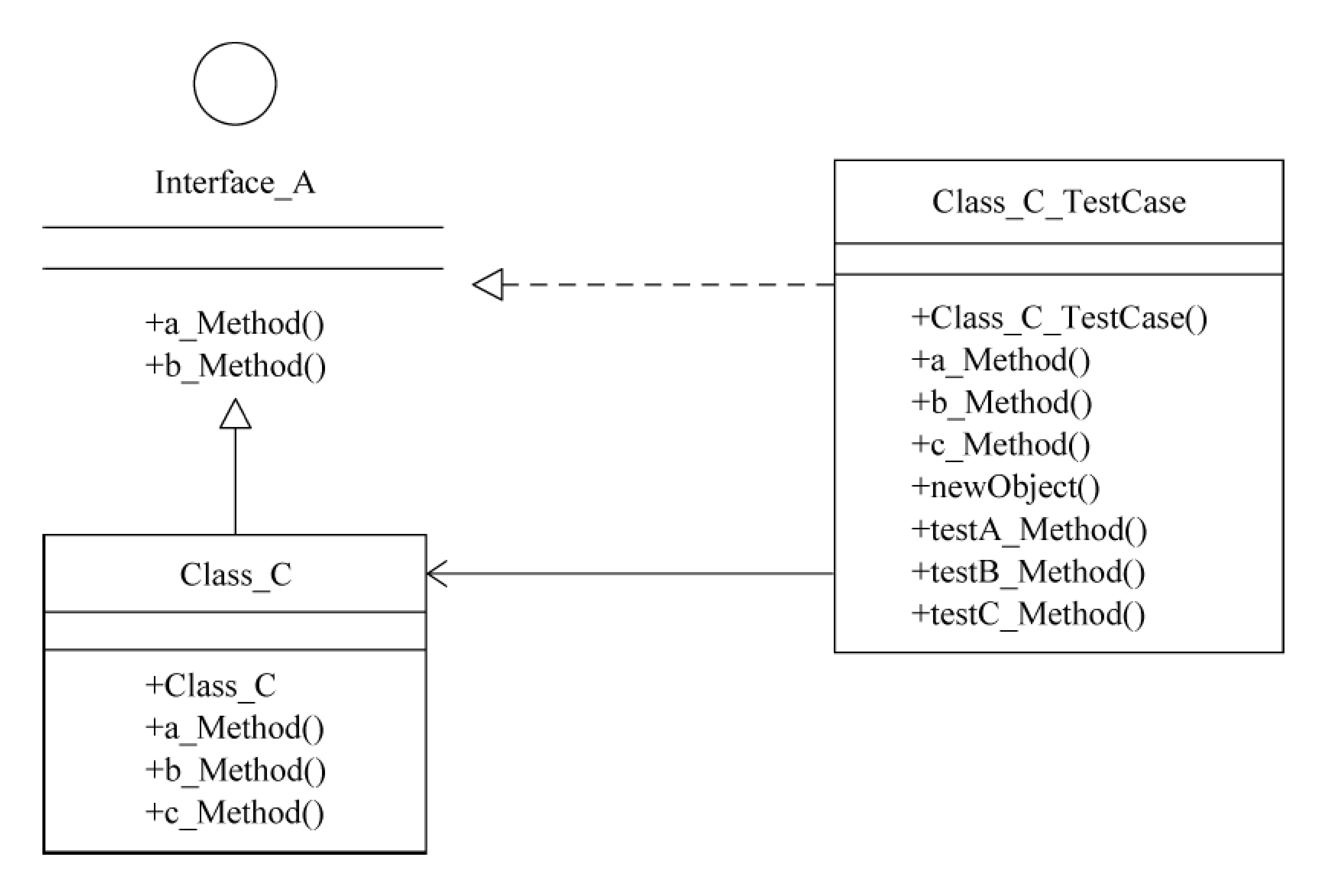


图 8-3　Interface 接口测试类图

3. 抽象类测试

由于使用抽象类可以提高开发和维护程序的效率，因此可以看到大量的抽象类被应用。但抽象类不能被实例化，而基于类的测试需要构建可执行的类实例。如果为抽象类单独开发一个具体子类来进行测试，就要额外花费一些时间，浪费宝贵的人力资源。那么，如何测试抽象类呢？如果要构造抽象类的测试驱动程序，首先要继承测试驱动类，并且需要同时继承被测试抽象类，因为该类不能被具体化。但是 Java 采用单继承机制，因此对该抽象类的测试驱动程序就不能同时继承两个抽象类，这似乎使问题的解决陷入了困境。其实，可以换个角度来考虑问题。通常，采用以下两种方法来处理抽象类的测试问题。

(1) 一般情况下，利用 Java 的内类机制，在抽象类的测试驱动程序内引入内类，让内类实现对被测试抽象类的继承，然后把它作为引用体，这样对内类的测试就等价于对被测试抽象类的测试。举例如下：

```
/*
AbstractExample.java
```

```
* 创建日期：
* 创建人员：
* 修改日期：
* 修改人员：
* /
package applet. unedu. tlpe. test;
public abstract class AbstractExample implements Example{
…
public AbstractExample(){
…
}
…
}

/ *
AbstractExampleTester. java
* 创建日期：
* 创建人员：
* 修改日期：
* 修改人员：
* /
package applet. unedu. tlpe. test;
public abstract class AbstractExampleTester extends TestCase{
static class AbstractExample_Inner extends AbstractExample{
public AbstractExample_Inner(){
super(argumentname);
}
}
public AbstractExampleTester(){
super(argumentname);
}
public Example newObject(argumentname){
return new AbstractExample_Inner();
}
public void testAbstractExample_Inner(){
…
}
…
}
```

（2）如果抽象类被具体类继承，那么，在创建该具体类的测试驱动程序时要继承抽象类的测试驱动程序，在以后的回归测试中，只要执行最底层的测试类，就可以对其父测试类重新执行一次测试，同时将测试结果分别返回。

4. 重载和覆盖测试

覆盖是在子类中重新定义了从父类中继承的同名方法；重载与覆盖不同，不是子类对父类同名方法的重新定义，而是类对自身已有的同名方法的重新定义。那么，对含有重载和覆盖方法的类如何进行测试呢？

在测试过程中，可以参考如下两个原则：

（1）要对类实例方法的所有重载形式分别进行测试。

（2）子类的测试驱动程序在继承父类测试驱动程序的同时，要对覆盖了父类的同名方法进行测试，而且应该重新对父类的类实例方法的所有重载形式执行一次测试。

举例如下：

```
package applet.unedu.tlpe.test;
public class reload{
public reload(){
}
public int method_a(){
return "method_a()";
}
public int method_a(String a){
return "method_a(String a)";
}
public int method_a(String a,String b){
return "method_a(String a,String b)";
}
public int method_b(){
return "method_b()";
}
public int method_c(){
return "method_c()";
}
}
```

上面程序代码定义了 reload 类，下面程序代码定义了 reload_TestCase 类，reload_TestCase 是 reload 类的测试驱动程序，该类对 reload 类中包含的 method_a 实例方法的 3 种重载形式分别进行了测试，代码如下：

```
/ *
reload_TestCase. java
 * 创建日期:
 * 创建人员:
 * 修改日期:
 * 修改人员:
 * /
package applet. unedu. tlpe. test;
public class reload_TestCase extends TestCase{
…
public reload_TestCase(argument){
…
}
public void testMethod_a(){…}
public void testMethod_a_a(){…}
public void testMethod_a_ab(){…}
public void testMethod_b(){…}
public void testMethod_c(){…}
…
}
```

如果现在有另外一个类 reload01 继承了 reload 类,并且覆盖了 reload 类中的同名方法 method_b()和 method_c(),具体代码略。那么首先让 reload01_TestCase 类继承 reload_TestCase 类,在 reload01_TestCase 类中要重新覆盖 testMethod_b()和 testMethod_c()两个方法;然后,reload01_TestCase 类在实现了对 reload01 类测试的基础上,也能够重新对 reload 类的重载(method_a()方法的 3 种重载形式)执行一次测试,具体代码略。

5. 异常测试

为了能够有效地处理程序运行中的错误,Java 中引入了异常和异常类,当执行体发生不可预料的结果时就会启动这种异常处理机制。因此,在构建测试用例时,除了要考虑怎样测试正常情况下的程序运行状态外,还要更好地考虑如何测试在某种特殊情况下程序代码是否返回了指定的状态。在测试要抛出异常的方法时要设计两类测试用例,即异常测试用例和正常测试用例。

8.5 类测试

面向对象程序的基本单位是类。类测试是由那些与验证类的实现是否和该类的说明完全一致的相关联活动组成的。如果类的实现正确,那么类的每一个实例的行为也应该

是正确的。

1. 类测试的方法

通过代码检查或执行测试用例能有效地测试一个类的代码。在某些情况下，用代码检查代替基于执行的测试方法是可行的。但是和基于执行的测试方法相比，代码检查有两个不利之处：

- 代码检查容易受人为错误的影响。
- 代码检查在新产品开发时明显需要更多的工作量。

尽管基于执行的测试方法克服了这些缺点，但确定测试和开发测试驱动程序也需要很大的工作量。在某些情况下，为某个类构造一个测试驱动程序所需要的工作量可能比开发这个类所需要的工作量还要大。但这种情况不是面向对象编程独有的，当有许多子程序被上一层次调用时，在传统开发过程中，也会出现类似的情况。一旦确定了一个类的可执行测试用例，测试驱动程序创建一个或多个类的实例来运行一个测试用例，我们就必须执行测试驱动程序来运行每个测试用例，并给出每个测试用例运行的结果。

2. 类测试的组成部分

作为每个类，决定是将其作为一个单元进行独立测试，还是以某种方式将其作为系统某个较大部分的一个组件进行独立测试，需要基于以下因素进行决策：

- 这个类在系统中的作用，尤其是与之相关联的风险程度。
- 这个类的复杂性(根据状态个数、操作个数以及关联其他类的程度等进行衡量)。
- 开发这个类测试驱动程序所需要的工作量。

假如一个类是某个类库不可缺少的部分，尽管测试驱动程序的开发成本可能很高，对它进行充分的测试也是值得的，因为它的正确操作是最重要的。在进行类测试时，一般要考虑以下几个方面：

- 测试人员。如同传统的单元测试是由开发人员来执行，类的测试通常也由开发人员来进行。因为测试人员对代码相当熟悉，开发人员可以使用测试驱动程序来调试他们编写的代码，方便了基于执行的测试方法。
- 测试内容。对一个类进行测试以检查它是否只做了规定的事情，确保一个类的代码能够完全满足类说明所描述的要求。在运行了各种测试用例后，如果代码的覆盖率不完整，这可能意味着这个类的设计过于复杂，需要简化成几个子类。
- 测试时间。类测试可以在开发过程的不同位置进行。在一个递增的、反复的开发过程中，一个类的说明/实现可能会发生变化，所以应该在软件其他部分使用该类之前来执行类的测试。每当一个类的实现发生变化时，就应该执行回归测试。

所以类的测试要和类的设计、开发保持同步，因为确定早期测试用例有助于对类说明的理解保持一致，也有助于获得独立代码检查的反馈。若一个类的开发人员不能设计出充分和准确的测试用例，其测试结果会给人一个错觉——即该类通过了所有测试。但是当该类被集成到某个较大的系统时，将会导致严重的问题。

- 测试过程。类的测试通常要借助测试驱动程序，这个驱动程序创建类的实例，并为那些实例创造适当的环境以便运行一个测试用例。驱动程序向测试用例指定的一个实例发送一个或多个消息，然后根据参数、响应值、实例发生的变化来检查那些消息产生的结果。如果编程语言(例如 C++)具有程序员管理存储分配的机制，那么测试驱动程序需要删除它所创建的那些实例。
- 测试程度。可以根据已经测试了多少类实现和多少类说明来衡量测试的充分性。对于类测试来说，要测试操作和状态转换的各种组合情况，但有时穷举法是不可能的，此时就应该选择配对系列的组合情况，如果能结合风险分析进行选择，效果就会明显些。

3. 构建测试用例

首先看怎样从类说明中确定测试用例，然后根据类实现引进的边界值来扩充附加的测试用例。类说明通常可以用多种方式进行描述，如自然语言和状态图等，假如要测试的类的说明不存在，那么就可通过“逆向工程法”产生一个说明，并在开始测试之前让开发人员对之进行检查。

根据前置条件和后置条件来构建测试用例的总体思想是：为所有可能出现的组合情况确定测试用例需求。在这些可能出现的组合情况下，可以满足前置条件，也能够达到后置条件。接下来创建测试用例来表达这些需求，根据这些需求还可以创建拥有特定输入值(包括常见值和边界值)的测试用例，并确定它们的正确输出，最后，还可以增加测试用例来阐述违反前置条件所发生的情况。

4. 类测试系列的充分性

在某些情况下，可以使用穷举法来测试每个类，即用所有可能的值来测试，以确保每一个类都符合它的说明。在这种情况下，穷举测试法所带来的好处就超过了编写测试驱动程序以运行更多测试用例所付出的代价。

但是穷举测试法一般是不可能实现的，如果不能使用穷举测试法时，就不能保证一个类的每一方面都符合它的说明，但能够运用某个充分性的标准来使我们对测试系列的质量抱有高度的信心。充分性的三个常用标准是：基于状态的覆盖率、基于约束的覆盖率、基于代码的覆盖率。最低限度地符合这些标准将会产生若干不同的测试系列。将所有三个标准用于测试系列，将会提高我们对测试充分性的信任度。

- 基于状态的覆盖率。以测试覆盖了多少个状态转换为依据。假如测试没有覆盖一个或一个以上的状态转换，那么这个类的测试就不充分。即使测试用例对所有的状态都覆盖了一次，测试的充分性仍值得怀疑，因为状态通常包含了各种对象属性的值域。这样，必须测试这些值域里的所有值，包括典型值和边界值。
- 基于约束的覆盖率。与基于状态转换的充分性类似，还可以根据有多少对前置条件和后置条件被覆盖来表示充分性。例如，如果一个操作的前置条件是 pre1 或者 pre2，而后置条件是 post1 或者 post2，充分的测试则需要包含所有有效的组合情况（即 pre1＝true，pre2＝false，post1＝true，post2＝false；pre1＝false，pre2＝true，post1＝true，post2＝false 等）的测试用例。假如生成的测试用例满足了每一个需求，那么就符合这个充分性的标准。
- 基于代码的覆盖率。当所有的测试用例都执行结束时，确定实现一个类的每一行代码，或代码通过的每一条路径至少执行了一次，这是一种很好的思想。即使代码覆盖率是 100%，也不一定能满足基于状态覆盖率或基于约束的覆盖率是 100%。因为基于代码的覆盖率不够充分，所以使用哪些度量标准中的某一种来确定充分性是很重要的。

5. 构建测试的驱动程序

测试驱动程序是一个运行测试用例并收集运行结果的程序。测试驱动程序的设计应该相对简单，因为实际工作中很少有时间和资源来对驱动程序软件进行基于执行的测试（否则会进入一个程序测试的递归的、无穷之路），而是依赖代码检查来检测测试驱动程序。所以，测试驱动程序必须是严谨的，结构清晰、简单，易于维护，并且对所测试类的说明变化具有很强的适应能力。理想情况下，在创建新的测试驱动程序时，应该能够复用已存在的驱动程序代码。

6. Tester 类的设计

由于 Tester 类提供了一些操作来帮助给出测试用例的结果，一个具体的 Tester 类的主要任务就是运行测试用例和给出结果。类接口的主要组成部分是建立测试用例的操作、分析测试用例结果的操作、执行测试用例的操作和创建用于运行测试用例的输出实例的操作。在具体的 Tester 类中，为每一个测试用例定义了一个方法，被称为测试用例方法。这些方法给测试计划提供了可跟踪性——每一个测试用例或每一组紧密联系的测试用例都有一个方法。测试用例方法的目的是通过创建输入状态、生成事件序列并检查输出状态来执行测试用例。

测试用例方法的任务是为某个用例构建输入状态。例如，可以通过将一个输出和作为参数传递的对象实例化，然后生成测试用例指定的事件。如图 8-4 所示显示了一个满足了这些需求的 Tester 类的模型。

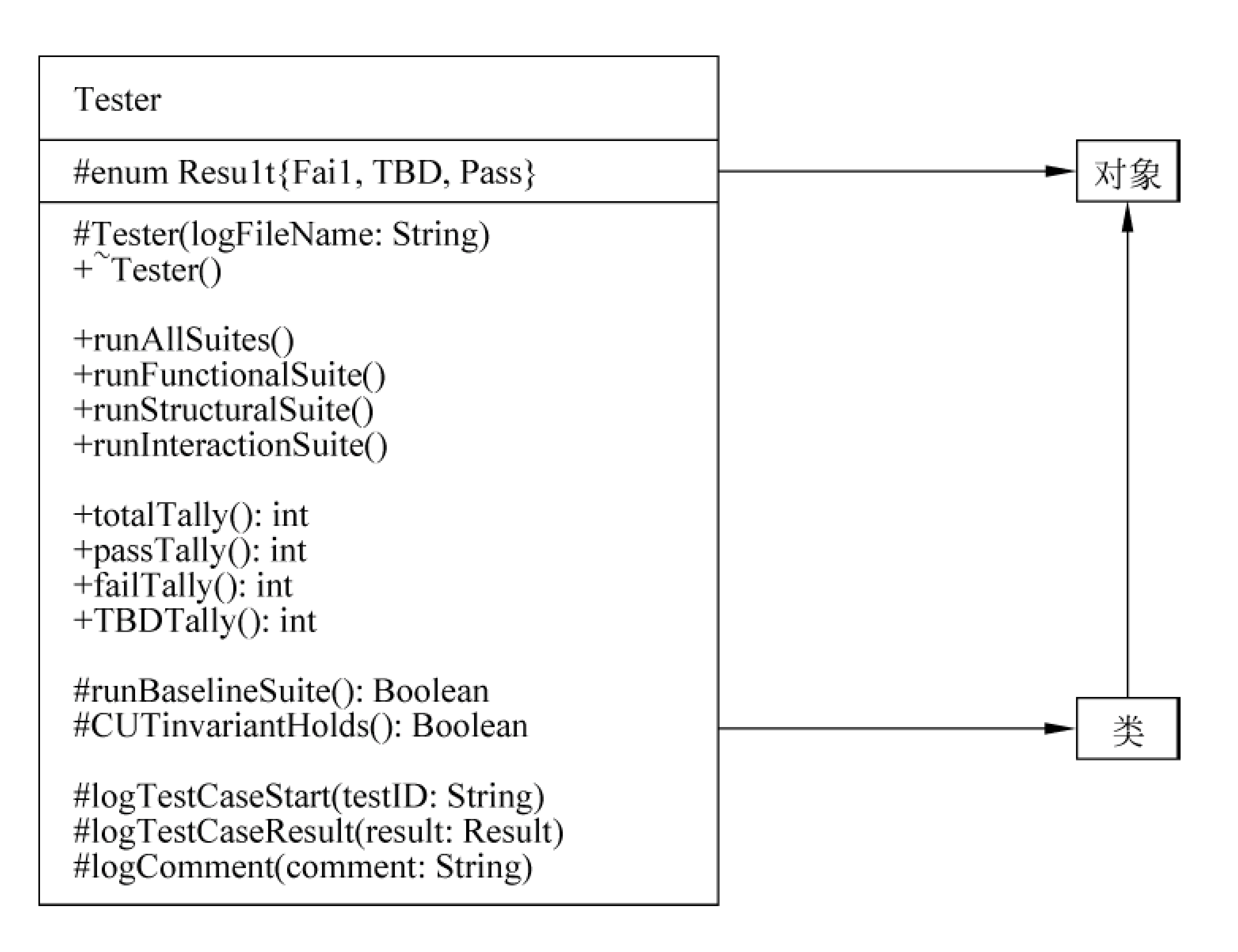


图 8-4 Tester 类需求的一个类模型

公共接口提供了一些操作来运行各种不同的测试,根据测试用例的来源组织测试系列。如果测试用例是根据类说明确定的,那么就是功能性测试用例;如果测试用例是根据代码确定的,那么就是结构性测试用例;如果测试用例是测试一个事件序列对一个对象的操作(例如几对输入输出转换是否正确),那么就是交互性测试用例。根据测试用例的依据来确定这些范畴是为了便于测试的维护。

设计测试用例的方法有多种,下面将简单介绍根据前置和后置条件确定测试用例的方法。

根据操作的前置和后置条件来确定测试用例的总体思想是:为所有可能出现的组合情况确定测试用例需求,然后创建测试用例来表达这些需求,最后排除不可能出现的情况。在实际的测试过程中,可以根据这些需求确定特定的输入值和输出值,还可以增加测试用例来描述违反前置条件时的情况。

为了便于从前置条件和后置条件中确定测试用例的总体需求,可以使用 OCL(对象约束语言)来分析每种逻辑关系,对使用不同种逻辑关系表示的前置条件和后置条件,分别列出相应的测试用例。表 8-2 和表 8-3 分别列出了因使用各种不同逻辑表达式表示前置条件和后置条件时产生的测试用例需求,其中表 8-2 中的粗体表示的是一些暗中使用了保护性设计而产生的额外测试用例。

接下来就可以使用前置条件和后置条件的各种组合情况来测试一个指定的操作,使用以上两张表来确定测试用例需求的步骤如下:

表 8-2　根据前置条件确定测试用例

逻辑表达式	影　　响
true	(true,post)
①	(①,post)
	(not ①,exception)
not ①	(not ①,post)
	(① and ②,post)
① and ②	(① and ②,post)
	(① and not ②,exception)
	(not ① and ②,exception)
	(not ① and not ②,exception)
① or ②	(①,post)
	(②,post)
	(① and ②,post)
	(not ① or not ②,exception)
① xor ②	(① and not ②,post)
	(not ① and ②,post)
	(① and ②,exception)
	(not ① and not ②,exception)
① implies ②	(not ①,post)
	(②,post)
	(not ① and ②,post)
	(① and not ②,exception)
if ① then ② else ③ endif	(① and ②,post)
	(not ① and ②,post)
	(① and not ②,exception)
	(not ① and not ③,exception)

注：①、②、③代表 OCL 表示的组件。

(1) 确定操作的前置条件与表 8-2 中的哪个逻辑表达式相匹配，然后找出相对应的影响列表。

(2) 确定操作的后置条件与表 8-3 中的哪个逻辑表达式相匹配，然后找出相对应的影响列表。

(3) 根据前面得出的影响列表确定各种组合情况构成测试用例需求，即使用第一个列表中的每个输入约束条件代替第二个列表中的每个前置条件。

(4) 去掉生成的所有无意义的条件。例如：(shape＝rectangle)or(shape＝circle)将产生一个测试用例，在这个测试用例中，不可能同时满足(shape＝rectangle)and(shape＝circle)。

表 8-3 根据后置条件确定测试用例

后置条件	影 响
(1)	(pre,(1))
(1)and(2)	(pre,(1)and(2))
(1)or(2)	(pre,(1))
	(pre,(2))
	(pre,(1)and(2))
(1)xor(2)	(pre,(1)and(2))
	(pre,not(1)and(2))
(1)implies(2)	(pre,not(1)or(2))
if(1)then(2)else(3)endif	(pre and ☆,(2))
	(pre and not ☆,(3))

注：☆就是使得(1)为真的一个条件。

个别情况下，有的前置条件和后置条件比表中所列的情况更为复杂，如涉及 3 个转折词，那么此时就循环执行步骤(1)和步骤(2)。但通常在面向对象程序中的前置条件和后置条件都很简单。

下面是针对一个类的某个操作测试的例子。

```
public setID(int ID){
try{
if(conditionA or conditionB)
ID=b+c;
}
catch(SQLException e){
e. printStackTrace();
}
}
```

使用 OCL 对该操作的描述如下：

```
setID 所属的类名：setID(ID：int)
pre：conditionA or conditionB
post：ID=b+c
```

在这个方法中，①代表 conditionA，②代表 conditionB；(1)代表 ID=b+c。

根据前面所述的确定测试用例需求的步骤，可以把形式和形式的前置条件以及后置条件的表象组合起来就可以确定如下几个测试用例：

考虑前置条件对测试系列的影响确定的测试用例如下：

(①,(1))	(conditionA,ID=b+c)
(②,(1))	(conditionB,ID=b+c)
(① and ②,(1))	(conditionA and conditionB,ID=b+c)
(not ① or not ②,exception)	(not conditionA and not conditionB,exception)

那么在实际类中,针对 conditionA 和 conditionB 给变量赋值,就可以得到满足上述的测试用例需求,设计具体的测试用例;同理,当逻辑表达式为其他情况时的测试用例需求,也可以按照类似的方法来确定。但要注意的是:要从所确定的测试用例需求中排除不可能发生的情况。

8.6　JUnit 简介

1. JUnit 概述

JUnit 是一个对 Java 程序进行单元测试的测试框架,是 xUnit 在 Java 语言上的实现。xUnit 是专门进行单元测试的程序框架。JUnit 是 SourceForge 上一个开源软件,它的官方网站是:www.junit.org,项目网站是:http://sourceforge.net/projects/junit/。

JUnit 有以下特点:

- 提升程序代码的品质时,JUnit 测试仍允许更快速的编写程序。如果采用一个综合的测试系列,就可以在改变程序代码之后快速地执行多个测试。
- JUnit 使用简单。使用 JUnit 可以快速地编写测试并检测程序代码,并逐步随着程序代码的增长增加测试。测试是检查程序代码的完整性。
- JUnit 能够检验测试结果并立即提供回馈。JUnit 测试可以自动执行并且检查结果。当执行测试时,将获得简单且立即的回馈,比如测试是通过或失败,而不再需要人工检查测试结果的报告。
- JUnit 测试可以合成一个有层次的测试系列架构。JUnit 可以把测试组织成测试系列,这个测试系列可以包含其他的测试或测试系列。JUnit 测试的合成行为允许组合多个测试并自动的回归,从头到尾测试整个测试系列,也可以执行测试系列层级架构中任何一层的测试。
- 开发测试成本低。测试是检验要测试的程序代码并定义期望的结果。JUnit 测试框架提供自动执行测试的背景,并使这个背景成为其他测试集合的一部分。花费少量的测试投资便能够持续地获得回报。
- JUnit 测试提升软件的稳定性。对程序所作的测试越少,程序代码就越不稳定。
- JUnit 测试是用 Java 开发的。使用 Java 测试 Java 软件形成一个介于测试及程序代码间的无缝边界。在测试的控制下测试变成整个软件的扩充,同时程序代码可以被重整。

2. JUnit 的安装

(1) 首先要从网上下载最新的软件压缩包，本书中所有例子使用的是 3.8.1 版本。

(2) 解压 JUnit 压缩包。将 JUnit.zip 复制到工作目录(C:\work)下，直接解压缩即得到 C:\work \Junit3.8.1。这时可以在这个目录下看到以下内容：

Readme.html：JUnit 说明文件。

cpl-v10.html：License 文件。

JUnit.jar：包含 JUnit 所有的类库。

Src.jar：JUnit 的源代码包。

Doc 目录：JUnit 帮助文档的目录。其中包含以下两个目录。

- JavaDoc 目录：JUnit 的 API 说明文档目录；
- JUnit 目录：JUnit 自带的例子程序目录(包含生产代码和测试代码)。

(3) 修改 CLASSPATH(以 Windows 2000 Server 操作系统为例)。做法：右键单击"我的电脑"图标，依次选择"属性"→"高级"→"环境变量"，在"系统变量"的"变量"列表框中选择 CLASSPATH，然后选择"编辑"，在变量名输入框中加入 JUnit.jar 所在路径，如在本例中加入 C:\work \Junit3.8.1\ JUnit.jar，单击"确定"按钮即可(注意：使用"；"号和其他变量值分开)。

(4) 运行验证程序。进入到安装目录(C:\work \junit3.8.1)。

- 运行字符界面的 JUnit 工具：输入 java junit.textui.TestRunner junit.samples.AllTests，结果如图 8-5 所示。

```
C:\work\junit3.8.1>java junit.textui.TestRunner junit.samples.AllTests
........................................
........................................
.......................................
Time: 2.033

OK (119 tests)


C:\work\junit3.8.1>
```

图 8-5　JUnit 的字符界面

图 8-5 中一个点表示测试成功一个单元测试(否则是 E)，Time 表示测试总共花费的时间，OK 说明没有发现任何错误，119 表明一共进行了 119 个单元测试。

- 运行图形界面的 JUnit 工具：输入 java junit.swingui.TestRunner junit.samples.AllTests，结果如图 8-6 所示。

绿色进度条就是著名的"Green Bar"，当 Green Bar 出现的时候，说明设计的单元测试全部获得了成功，可以进行下一步了。

如果出现如图 8-5 和图 8-6 所示的界面时，说明已经成功安装了 JUnit。

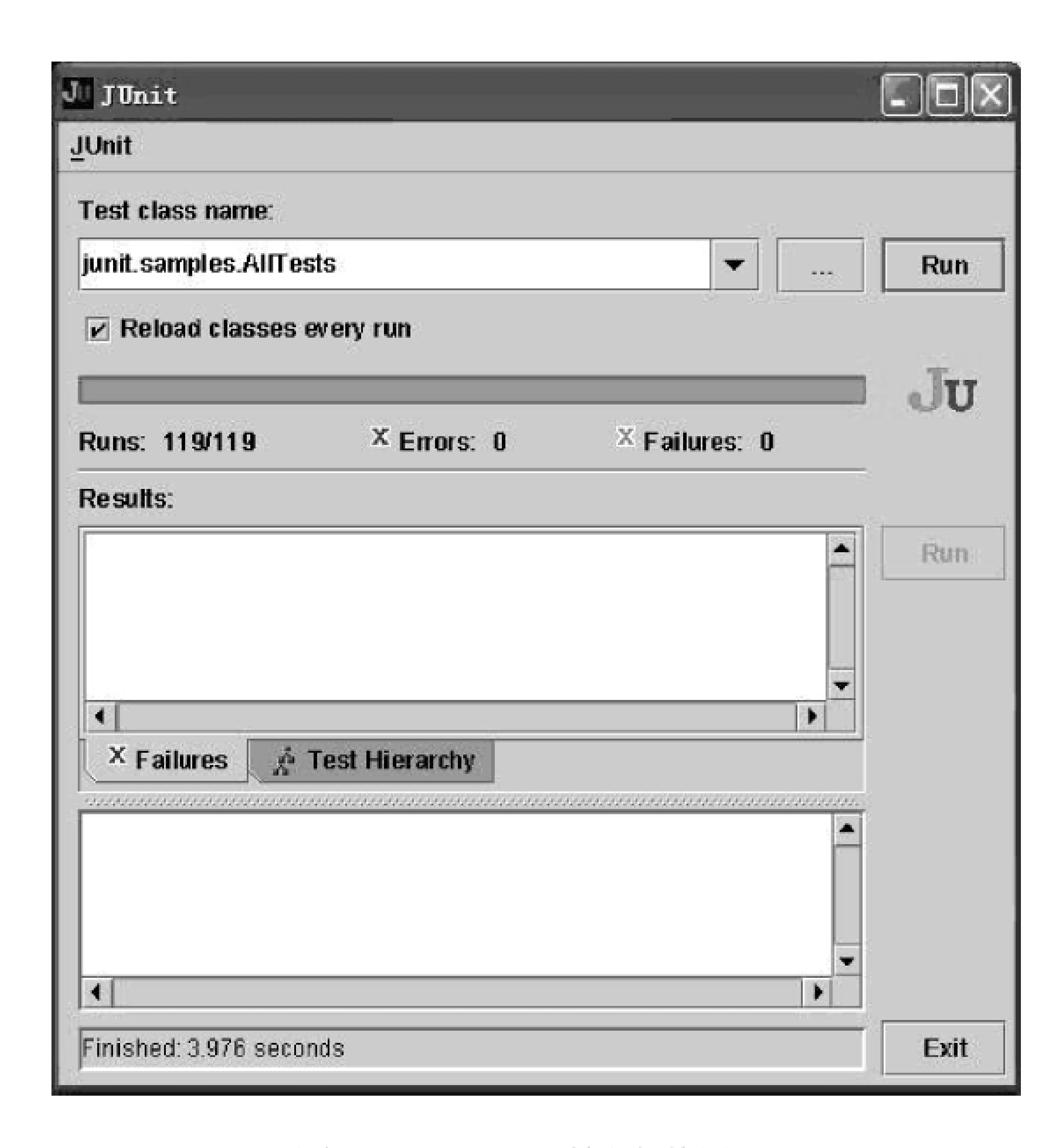


图 8-6　JUnit 的图形界面

3. JUnit 类概述

JUnit 有两个包：junit. framework 和 junit. extensions。junit. framework 是 JUnit 对 xUnit 框架的具体实现，是整个测试框架的核心。Junit. extensions 是 JUnit 测试框架的扩展。这里重点介绍 junit. framework 这个包。

junit. framework 包含 3 个接口：

- Protectable：JUnit 框架抛出异常的接口。
- Test：JUnit 框架中测试实例需要实现的接口。
- Testistener：对于 JUnit 的测试运行过程进行监听的 Listener 需要实现的接口。

junit. framework 包含 5 个类：

- Assert：断言类。这个类包含一系列方法实现单元测试级别的断言。
- TestCase：测试案例类。用户可以通过继承这个类来设置各种各样的测试案例。
- TestSuit：测试案例套装类。这个类是 TestCase 的一个集合，可以对测试案例进行组织和管理。
- TestFailer：测试失败类。这个类收集测试过程中各种异常和失败信息。
- TestResult：测试结果类。这个类收集测试的结果信息。

junit. framework 包含 2 个错误类：

- AssertionFailedError：断言失败错误类。当断言失败时，抛出这个类。
- ComparisonFailure：比较失败类。这个类是对 AssertionFailedError 的扩展。当比较字符串失败时，抛出这个类。

4. 使用 JUnit 来测试一段简单的 Java 代码

本文中所有的例子均使用 Eclipse 开发。

测试需求：编写一个 Hello 类。

测试案例：是否存在 Hello 类。

测试/编码步骤：

(1) 在 Eclipse 中创建一个名为 Hello 的空白工程。在 Eclipse 中创建一个名为 TestHello 的 JUnit Test Case 类，如图 8-7 所示。

(2) 在 TestHello 类中增加 testHello 方法，测试 Hello 类是否存在，如图 8-8 所示。

(3) 运行 TestHello，出现了著名的 Red Bar——存在测试没有通过！如图 8-9 所示。

(4) 在工程中添加空的 Hello 类，如图 8-10 所示。

(5) 修改 TestHello. testHello() 方法，断言 hello 变量是否可以声明，如图 8-11 所示。

(6) 运行 TestHello，出现了著名的 Green Bar——测试全部通过！如图 8-12 所示。

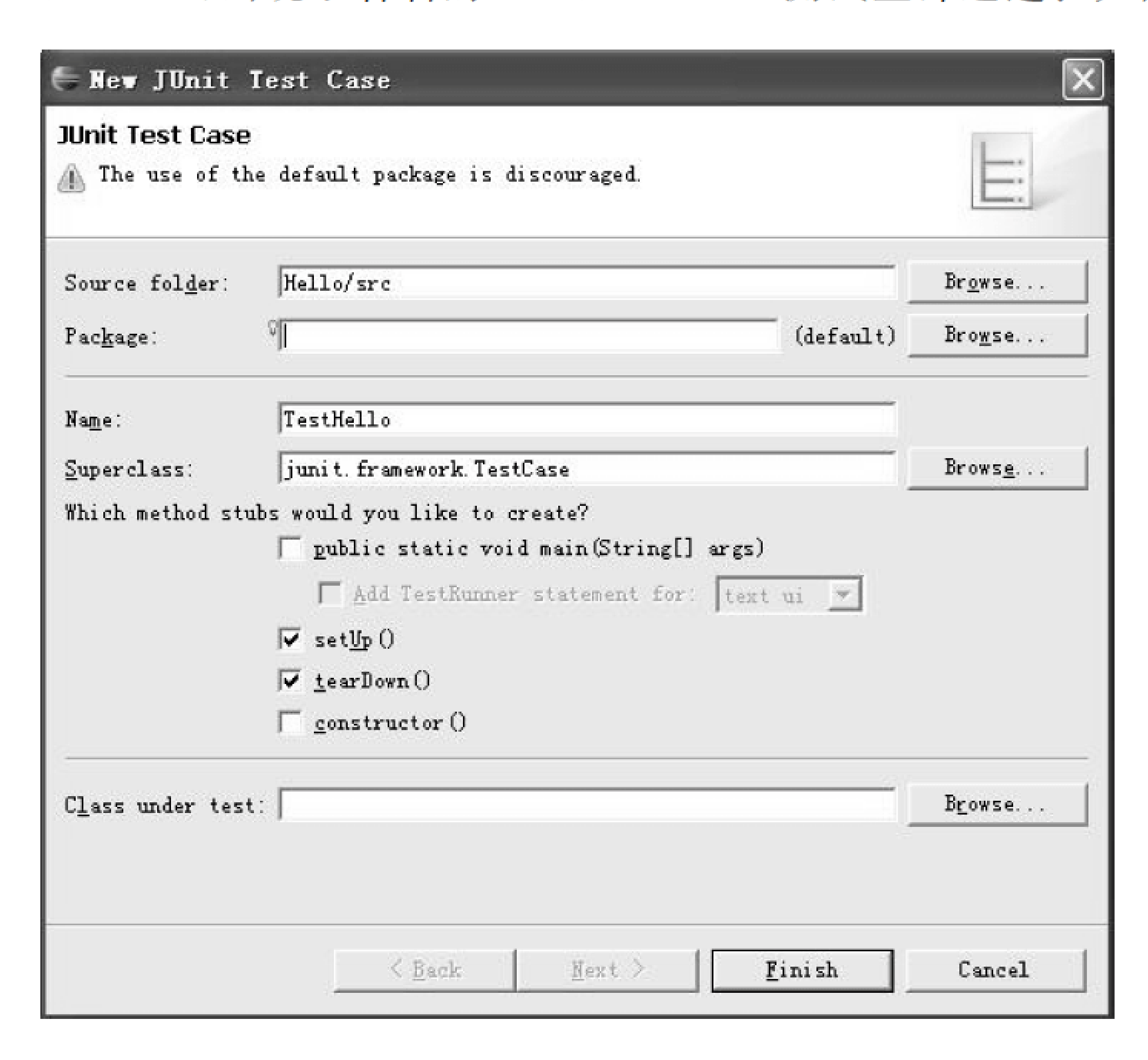


图 8-7 创建 TestHello 类

```
import junit.framework.TestCase;

public class TestHello extends TestCase {


    protected void setUp() throws Exception {
        super.setUp();


    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

    public void testHello()
    {
        throw new RuntimeException ("Implement me!");


    }


}
```

图 8-8　testHello()方法

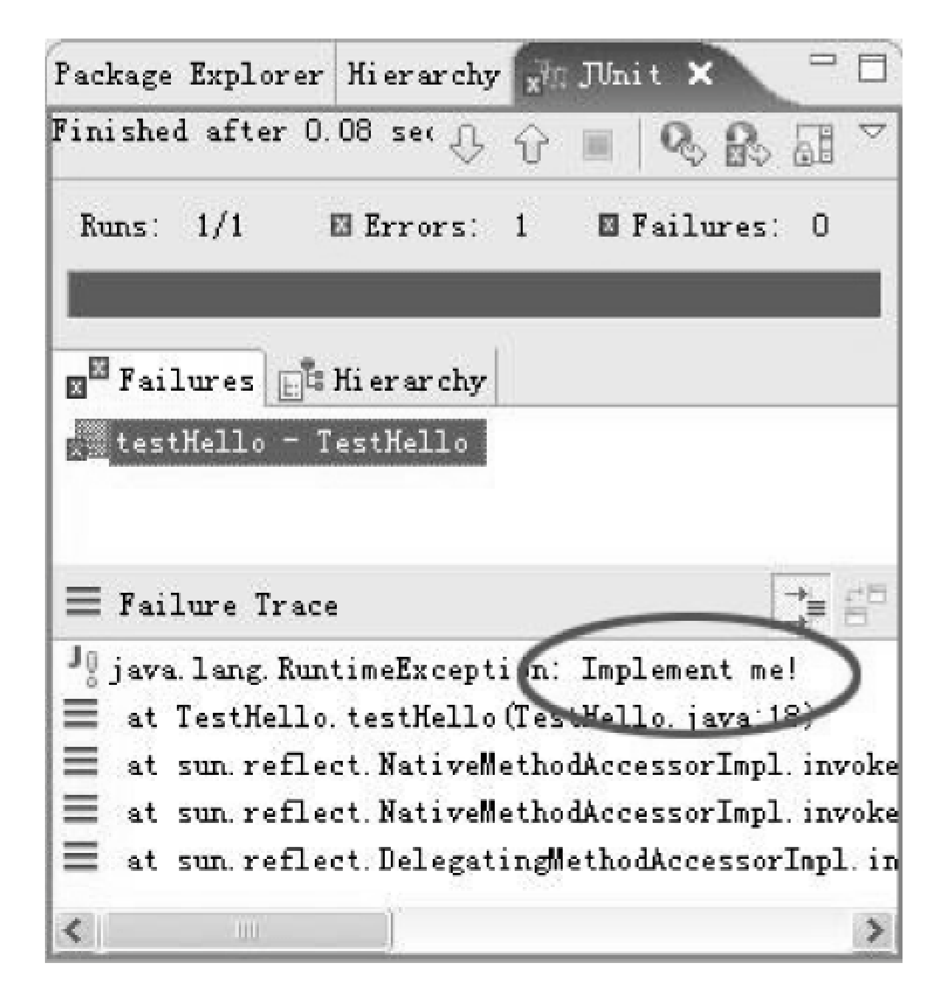


图 8-9　运行 TestHello，出现 Red Bar



图 8-10　添加空的 Hello 类

```java
import junit.framework.TestCase;

public class TestHello extends TestCase {

    private Hello hello;

    protected void setUp() throws Exception {
        super.setUp();

        hello = new Hello();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

    public void testHello()
    {
        //throw new RuntimeException ("Implement me!");

        assertNotNull(hello);
    }

}
```

图 8-11　断言 hello 变量

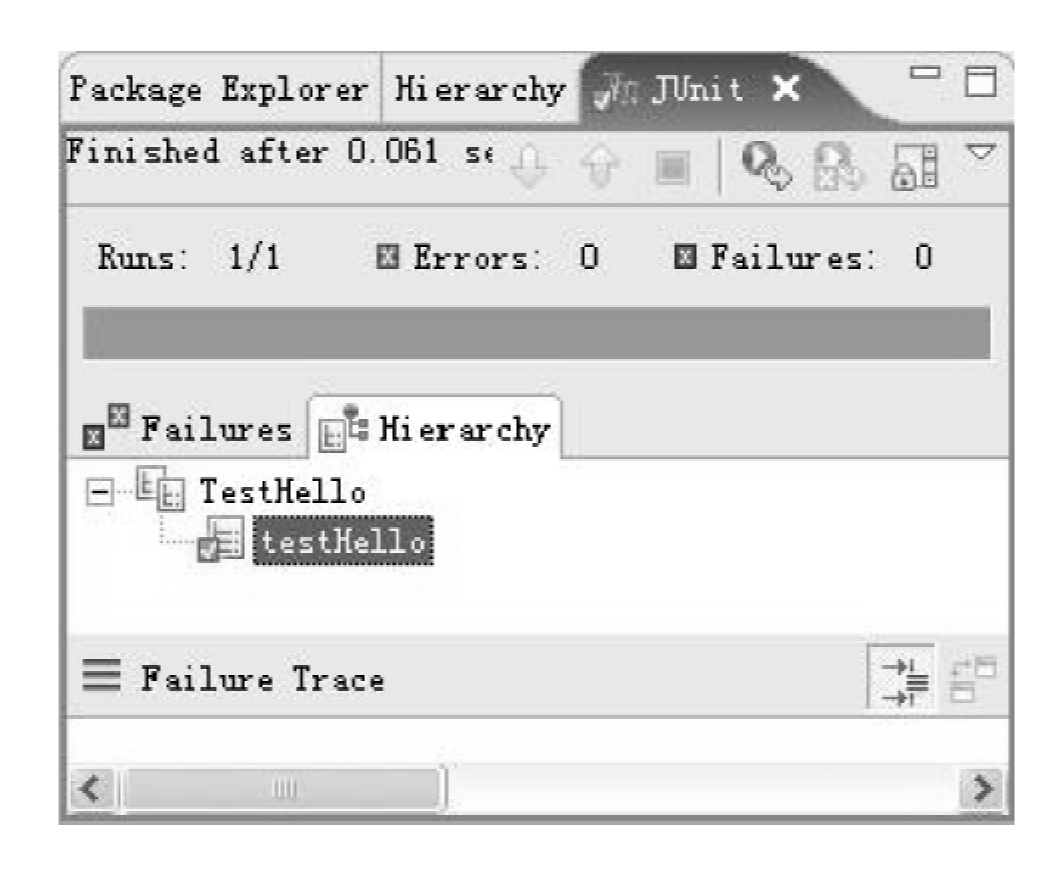


图 8-12　TestHello 测试通过

小　　结

本章在分析面向对象的软件测试与传统的软件测试的基础之上，对面向对象的软件测试的基本概念进行了简单的介绍；接下来分析了面向对象的软件测试技术，如：测试

用例和测试驱动程序的设计，测试抽象类和接口类，测试重载和覆盖等技巧。

习　　题

1. 简述面向对象的特点与其测试的概念。
2. 面向对象的软件测试与传统的软件测试有什么区别？
3. 通常情况下，类测试驱动程序是由测试人员还是程序员进行开发的？
4. 如何对抽象类进行测试？
5. 如何对接口类进行测试？
6. 如何测试重载和覆盖？

第 9 章

Web 网站测试

本章概述

Web 网站测试是面向因特网 Web 页面的测试。众所周知,因特网网页是由文字、图形、声音、视频和超级链接等组成的文档。网络客户端用户通过在浏览器中的操作,搜索浏览所需要的信息资源。

针对 Web 网站这一特定类型软件的测试,包含了许多测试技术,如功能测试、压力/负载测试、配置测试、兼容性测试、安全性测试等。黑盒测试、白盒测试、静态测试和动态测试都有可能被采用。

9.1 Web 网站的测试

随着因特网的快速发展和广泛应用,Web 网站已经应用到政府机构、企业公司、财经证券、教育娱乐等各个方面,对我们的工作和生活产生了深远的影响。正因为 Web 能够提供各种信息的连接和发布,并且内容易于被终端用户存取,使得其非常流行、无所不在。现在,许多传统的信息和数据库系统正在被移植到因特网上,复杂的分布式应用也正在 Web 环境中出现。

基于 Web 网站的测试是一项重要、复杂并且富有难度的工作。Web 测试相对于非 Web 测试来说是更具挑战性的工作,用户对 Web 页面质量有很高的期望。基于 Web 的系统测试与传统的软件测试不同,它不但需要检查和验证是否按照设计所要求的项目正常运行,而且还要测试系统在不同用户的浏览器端的显示是否合适。另外,还要从最终用户的角度进行安全性和可用性测试。然而,因特网和 Web 网站的不可预见性使测试基于 Web 的系统变得困难。因此,迫切需要研究基于 Web 网站的测试方法和技术。

针对 Web 的测试方法应该尽量覆盖 Web 网站的各个方面,测试技术方面在继承传统测试技术的基础上要结合 Web 应用的特点。

基于 Web 的系统测试与传统的软件测试既有相同之处,也有不同的地方,对软件测

试提出了新的挑战。基于 Web 的系统测试不但需要检查和验证是否按照设计的要求运行，而且还要评价系统在不同用户的浏览器端的显示是否合适。更需要从最终用户的角度进行安全性和可用性测试。

通常 Web 网站测试的内容包含以下方面：

- 功能测试；
- 性能测试；
- 安全性测试；
- 可用性/易用性测试；
- 配置和兼容性测试；
- 数据库测试；
- 代码合法性测试；
- 完成测试。

实际上 Web 网页各种各样，针对具体情况选用不同的测试方法和技术。例如，如图 9-1 所示是一个典型的 Web 网页，具有各种可测试特性。而如图 9-2 所示是一个简单的网站首页，界面直观，仅由简单的文字、图片和链接组成，测试起来并不困难。



图 9-1　一个典型的 Web 网页



图 9-2 一个简单的 Web 网页

本章将从功能测试、性能测试、安全性测试、可用性/易用性测试、配置和兼容性测试、数据库测试、代码合法性测试和完成测试几个方面讨论了基于 Web 的系统测试方法。

9.2 功能测试

功能测试是测试中的重点，在实际的测试工作中，功能在每一个系统中具有不确定性，而我们不可能采用穷举的方法进行测试。测试工作的重心在于 Web 站点的功能是否符合需求分析的各项要求。

对于网站的测试而言，每一个独立的功能模块都需要设计相应的测试用例进行测试。功能测试的主要依据为《需求规格说明书》及《详细设计说明书》。对于应用程序模块则要采用基本路径测试法的测试用例进行测试。

功能测试主要包括以下几个方面的内容：

- 内容测试；
- 链接测试；
- 表单测试；
- Cookies 测试；
- 设计语言测试。

9.2.1　页面内容测试

内容测试用来检测 Web 应用系统提供信息的正确性、准确性和相关性。

(1) 正确性

信息的正确性是指信息是真实可靠的还是胡乱编造的。例如，一条虚假的新闻报道可能引起不良的社会影响，甚至会让公司陷入麻烦之中，也可能引起法律方面的问题。

(2) 准确性

信息的准确性是指网页文字表述是否符合语法逻辑或者是否有拼写错误。在 Web 应用系统开发的过程中，开发人员可能不是特别注重文字表达，有时文字的改动只是为了页面布局的美观。可怕的是，这种现象恰恰会产生严重的误解。因此测试人员需要检查页面内容的文字表达是否恰当。这种测试通常使用一些文字处理软件来进行，例如使用 Microsoft Word 的“拼音与语法检查”功能。但仅仅利用软件进行自动测试是不够的，还需要人工测试文本内容。

另外，测试人员应该保证 Web 站点看起来更专业些。过分地使用粗斜体、大号字体和下划线可能会让人感到不舒服，一篇到处是大字体的文章会降低用户的阅读兴趣。

(3) 相关性

信息的相关性是指能否在当前页面内可以找到与当前浏览信息相关的信息列表或入口，也就是一般 Web 站点中所谓的“相关文章列表”。测试人员需要确定是否列出了相关内容的站点链接。如果用户无法点击这些地址，他们可能会觉得很迷惑。

页面文本测试还应该包括文字标签，它为网页上的图片提供特征描述。如图 9-3 所示给出的是网页中一个文字标签的例子。当用户把鼠标移动到网页的某些图片时，就会立即弹出关于图片的说明性语言。

大多数浏览器都支持文字标签的显示，借助文字标签，用户可以很容易地了解图片的语义信息。进行页面内容测试时，如果整个页面充满图片，却没有任何文字标签说明，那么会影响用户的浏览效果。

网上店面是现在非常流行的 Web 网站，这里设定一个网上小百货商店作为例子，并为其设计测试用例。

网上商店有多种商品类别供用户选择，用户选中商品后放入购物车。当选完商品，应用程序自动生成结账单，用户就可以进行网上支付、购买商品了。

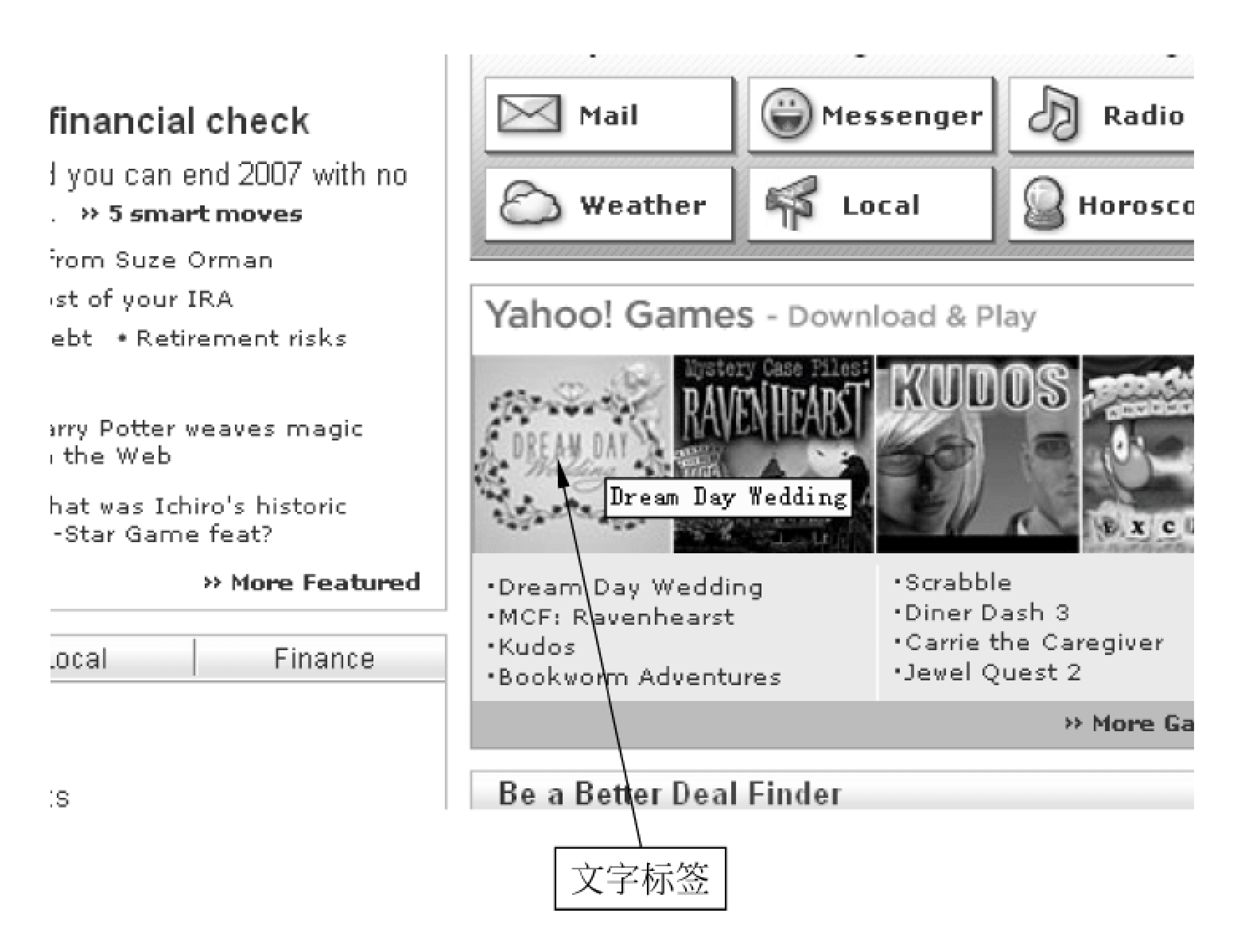


图 9-3 网页中的文字标签

本章除了数据库测试用例外，其他测试用例都是以网上商店为实例设计的，在下面的各小节中不再重复说明。

页面内容测试用例如表 9-1 所示。

表 9-1 页面内容测试用例示例

测试用例号	操作描述	数据	期望结果	实际结果
9.1	搜索某种类别的商品	搜索类别＝	搜索结果中列出该类别的所有商品	一致/不一致
9.2	让鼠标滑过每一个对象	受测对象＝	当鼠标滑过每一个对象时，显示相应的文本信息	一致/不一致

9.2.2 页面链接测试

链接是使用户可以从一个页面浏览到另一个页面的主要手段，是 Web 应用系统的一个主要特征，它是在页面之间切换和指导用户去一些不知道地址的页面的主要手段。链接测试需要验证三个方面的问题：

- 用户单击链接是否可以顺利地打开所要浏览的内容，即链接是否按照指示的那样确实链接到了要链接的页面。
- 所要链接的页面是否存在。实际上，好多不规范的小型站点，其内部链接都是空的，这让浏览者感觉很不好。
- 保证 Web 应用系统上没有孤立的页面，所谓孤立页面是指没有链接指向该页面，

只有知道正确的 URL 地址才能访问。

超级链接对于网站用户而言意味着能不能流畅地使用整个网站提供的服务，因而链接将作为一个独立的项目进行测试。另外，链接测试必须在集成测试阶段完成，也就是说，在整个 Web 应用系统的所有页面开发完成之后进行链接测试。

目前链接测试采用自动检测网站链接的软件来进行，已经有许多自动测试工具可以采用。如 Xenu Link Sleuth，主要测试链接的正确性，但是对于动态生成的页面的测试会出现一些错误。

页面测试链接和界面测试中的连接不同，前者注重功能，后者更注重连接方式和位置。页面测试链接更注重是否有链接，链接的页面是否是说明的位置等。

9.2.3　表单测试

当用户给 Web 应用系统管理员提交信息时，就需要使用表单操作，例如用户注册、登录、信息提交等。表单测试主要是模拟表单提交过程，检测其准确性，确保每一个字段在工作中正确。

表单测试主要考虑以下几个方面的内容：

- 表单提交应当模拟用户提交，验证是否完成功能，如注册信息。当用户通过表单提交信息的时候，都希望表单能正常工作。如果使用表单来进行在线注册，要确保提交按钮能正常工作，注册完成后应返回注册成功的消息。
- 要测试提交操作的完整性，以校验提交给服务器的信息的正确性。例如，个人信息表中，用户填写的出生日期与职称是否恰当，填写的所属省份与所在城市是否匹配等。如果使用了默认值，还要检验默认值的正确性。如果表单只能接受指定的某些值，则也要进行测试。例如，只能接受某些字符，测试时可以跳过这些字符，看系统是否会报错。
- 使用表单收集配送信息时，应确保程序能够正确处理这些数据。要测试这些程序，需要验证服务器能正确保存这些数据，而且后台运行的程序能正确解释和使用这些信息。
- 要验证数据的正确性和异常情况的处理能力等，注意是否符合易用性要求。
- 在测试表单时，会涉及数据校验问题。如果根据已定规则需要对用户输入进行校验，需要保证这些校验功能正常工作。例如，省份的字段可以用一个有效列表进行校验。在这种情况下，需要验证列表完整而且程序正确调用了该列表（例如在列表中添加一个测试值，确定系统能够接受这个测试值）。

提交数据，处理数据等如果有固定的操作流程可以考虑自动化测试工具的录制功能，编写可重复使用的脚本代码，这样就可以在测试、回归测试时运行以便减轻测试人员工作量。

如图 9-4 所示则是一个比较复杂的表单例子，用户填写个人信息，提交后可以申请 YAHOO 的免费信箱。

Yahoo! Registration - Microsoft Internet Explorer

文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)

后退 搜索 收藏夹 媒体

地址(D) https://edit.yahoo.com/config/eval_register?.intl=us&new=1&.done=http%3A//mail.yahoo.com&.src=ym&.v=0&.u=0v 转到 链接

YAHOO! MAIL Yahoo! - Help

Already have an ID or a Yahoo! Mail address? Sign In.

Fields marked with an asterisk * are required.

Create Your Yahoo! ID

* First name:

* Last name:

* Preferred content: Yahoo! U.S.

* Gender: [Select]

* Yahoo! ID: @yahoo.com

ID may consist of a-z, 0-9, underscores, and a single dot (.)

* Password:

Six characters or more; capitalization matters!

* Re-type password:

If You Forget Your Password...

* Security question: [Select a Question]

* Your answer:

Four characters or more. Make sure your answer is memorable for you but hard for others to guess!

* Birthday: [Select a Month] dd , yyyy

* ZIP/Postal code:

Alternate Email:

Verify Your Registration

* Enter the code shown: More info

This helps Yahoo! prevent automated registrations.

Terms of Service

Please review the following terms and indicate your agreement below. Printable Version

1. ACCEPTANCE OF TERMS
Yahoo! Inc. ("Yahoo!") welcomes you. Yahoo!
provides its service to you subject to the
following Terms of Service ("TOS"), which may be

By clicking "I Agree" you agree and consent to (a) the Yahoo! Terms of Service and Privacy Policy, and (b) receive required notices from Yahoo! electronically.

I Agree I Do Not Agree

Code verification technology developed in collaboration with the Captcha Project at Carnegie Mellon University.

Copyright © 2007 Yahoo! Inc. All rights reserved. Copyright/IP Policy Terms of Service

NOTICE: We collect personal information on this site.

To learn more about how we use your information, see our Privacy Policy

Internet

图 9-4 表单示例

表单测试用例如表 9-2 所示。

表 9-2　表单测试用例示例

测试用例号	操作描述	数　据	期望结果	实际结果
9.3	使用 Tab 键从一个字段区跳到下一个字段区	开始字段区＝	字段按正确的顺序移动	一致/不一致
9.4	输入字段所能接受的最长的字符串	字段名＝ 字符串＝	字段区能够接受输入	一致/不一致
9.5	输入超出字段所能接受的最大长度的字符串	字段名＝ 字符串＝	字段区拒绝接受输入的字符	一致/不一致
9.6	在某个可选字段区中不填写内容，提交表单	字段名＝	在用户正确填写其他字段区的前提下，Web 程序接受表单	一致/不一致
9.7	在一个必填字段区中不填写内容，提交表单	字段名＝	表单页面弹出信息，要求用户必须填写必填字段区的信息	一致/不一致

9.2.4　Cookies 测试

Cookies 通常用来存储用户信息和用户在某个应用系统的操作，当一个用户使用 Cookies 访问了某一个应用系统时，Web 服务器将发送关于用户的信息，把该信息以 Cookies 的形式存储在客户端计算机上，这可用来创建动态和自定义页面或者存储登录等信息。关于 Cookies 的使用可以参考浏览器的帮助信息。如果使用 B/S 结构 Cookies 中存放的信息更多。

如果 Web 应用系统使用了 Cookies，测试人员需要对它们进行检测。测试的内容可包括 Cookies 是否起作用，是否按预定的时间进行保存，刷新对 Cookies 有什么影响等。如果在 Cookies 中保存了注册信息，请确认该 Cookies 能够正常工作而且已对这些信息加密。如果使用 Cookies 来统计次数，需要验证次数累计正确。

Cookies 测试用例示例如表 9-3 所示。

表 9-3　Cookies 测试用例示例

测试用例号	操作描述	数　据	期望结果	实际结果
9.8	测试 Cookies 打开和关闭状态	Web 网页＝	Cookies 在打开时是否起作用	一致/不一致

9.2.5 设计语言测试

Web 设计语言版本的差异可以引起客户端或服务器端的一些严重问题，例如使用哪种版本的 HTML 等。当在分布式环境中开发时，开发人员都不在一起，这个问题就显得尤为重要。除了 HTML 的版本问题外，不同的脚本语言，例如 Java、JavaScript、ActiveX、VBScript 或 Perl 等也要进行验证。

9.2.6 功能测试用例

功能测试用例如表 9-4 所示。

表 9-4 功能测试用例示例

测试用例号	操作描述	数据	期望结果	实际结果
9.9	1. 进入商品目录列表所在的页面 2. 选择若干商品并将它们添加到购物车中 3. 查看购物车	添加的商品＝ 购物车＝ 页面＝	购物车中列出所有选择的商品	一致/不一致
9.10	1. 通过搜索，选择不同网页中的商品，添加到购物车中 2. 查看购物车	添加的商品＝ 购物车＝ 搜索的关键词＝ 页面＝	购物车中列出所有选择的商品	一致/不一致
9.11	选择商品但没有放到购物车中	添加的商品＝ 购物车＝	购物车中没有所选中的商品	一致/不一致
9.12	1. 选择一些商品并把它们放到购物车中 2. 不查看购物车 3. 转到结账处	添加的商品＝ 购物车＝ 结账单＝	放到购物车中的商品在结账单中显示	一致/不一致
9.13	1. 选择一些商品并把它们放到购物车中 2. 把其中一件商品从购物车中取走	添加的商品＝ 取出的商品＝ 购物车＝	购物车中的商品随时更新以反映商品的添加和取出	一致/不一致

9.3 性能测试

网站的性能测试对于网站的运行非常重要，目前多数测试人员都很重视对于网站的性能测试。

网站的性能测试主要从三个方面进行：负载测试、压力测试和连接速度测试。

负载测试指的是进行一些边界数据的测试。压力测试更像是恶意测试。压力测试倾

向应该是致使整个系统崩溃。连接速度测试指的是打开网页的响应速度测试。

9.3.1　负载测试

性能测试需要验证 Web 系统能否在同一时间响应大量的用户，在用户传送大量数据的时候能否响应，系统能否长时间运行。可访问性对用户来说是极其重要的。如果用户得到“系统忙”的信息，他们可能放弃，并转向竞争对手。这样就需要进行负载测试。

负载测试是为了测量 Web 系统在某一负载级别上的性能，以保证 Web 系统在需求范围内能正常工作。负载级别可以是某个时刻同时访问 Web 系统的用户数量，也可以是在线数据处理的数量。

负载测试包括的问题有：Web 应用系统能允许多少个用户同时在线；如果超过了这个数量，会出现什么现象；Web 应用系统能否处理大量用户对同一个页面的请求。

负载测试的作用是在软件产品投向市场以前，通过执行可重复的负载测试，预先分析软件可以承受的并发用户的数量极限和性能极限，以便更好地优化软件。

负载测试应该安排在 Web 系统发布以后，在实际的网络环境中进行测试。因为一个企业的内部员工，特别是项目组人员总是有限的，而一个 Web 系统能同时处理的请求数量将远远超出这个限度，所以，只有放在因特网上，接受负载测试，其结果才是正确可信的。

Web 负载测试一般使用自动化工具来进行。

9.3.2　压力测试

系统检测不仅要使用户能够正常访问站点，在很多情况下，可能会有黑客试图通过发送大量数据包来攻击服务器。出于安全的原因，测试人员应该知道当系统过载时，需要采取哪些措施，而不是简单地提升系统性能。这就需要进行压力测试。

进行压力测试是指实际破坏一个 Web 应用系统，测试系统的反映。压力测试是测试系统的限制和故障恢复能力，也就是测试 Web 应用系统会不会崩溃，在什么情况下会崩溃。黑客常常提供错误的数据负载，通过发送大量数据包来攻击服务器，直到 Web 应用系统崩溃，接着当系统重新启动时获得存取权。无论是利用预先写好的工具，还是创建一个完全专用的压力系统，压力测试都是用于查找 Web 服务（或其他任何程序）问题的本质方法。

压力测试的区域包括表单、登录和其他信息传输页面等。

负载/压力测试应该关注的问题如下。

1. 瞬间访问高峰

例如电视台的 Web 站点，如果某个收视率极高的电视选秀节目正在直播并进行网上投票，那么最好使系统在直播的这段时间内能够响应上百万上千万的请求。负载测试工具能够模拟 X 个用户同时访问测试站点。

2. 每个用户传送大量数据

例如网上购物过程中，一个终端用户一次性购买大量的商品。或者节日里，一个客户网

上派送大量礼物给不同的终端用户等。系统都要有足够能力处理单个用户的大量数据。

3. 长时间的使用

Web 站点提供基于 Web 的 E-mail 服务具有长期性，其对应的测试就属于长期性能测试，可能需要使用自动测试工具来完成这种类型的测试，因为很难通过手工完成这些测试。你可以想象组织 100 个人同时点击某个站点。但是同时组织 100000 个人就很不现实。通常，测试工具在第二次使用的时候，它创造的效益，就足以支付成本。而且，测试工具安装完成之后，再次使用的时候，只要点击几下。

负载/压力测试需要利用一些辅助工具对 Web 网站进行模拟测试。例如，模拟大的客户访问量，记录页面执行效率，从而检测整个系统的处理能力。目前常用的负载/压力测试工具有 WinRunner、LoadRunner、Webload 等，运用它们可进行自动化测试。

9.3.3 连接速度测试

连接速度测试是对打开网页的响应速度测试。

用户连接到 Web 应用系统的速度根据上网方式的变化而变化，他们或许是电话拨号，或是宽带上网。当下载一个程序时，用户可以等较长的时间，但如果仅仅访问一个页面就不会这样。如果 Web 系统响应时间太长（例如超过 10 秒钟），用户就会没有耐心等待而离开。

另外，有些页面有超时的限制，如果响应速度太慢，用户可能还没来得及浏览内容，就需要重新登录了。而且，连接速度太慢，还可能引起数据丢失，使用户得不到真实的页面。

连接速度测试用例如表 9-5 所示。

表 9-5 连接速度测试用例示例

测试用例号	操作描述	数据	期望结果	实际结果
9.14	1. 提交一个完整的购买表单 2. 记录接收到购买确认的响应时间 3. 重复上述操作 5 次	购买的商品＝	记录最小、最大和平均响应时间，同时满足系统的性能要求	一致/不一致
9.15	1. 查找一件商品 2. 记录查找的响应时间 3. 重复上述操作 5 次	查询＝	记录最小、最大和平均响应时间，同时满足系统的性能要求	一致/不一致

9.4 安全性测试

随着因特网的广泛使用，网上交费、电子银行等深入到了人们的生活中。所以网络安全问题就日益重要，特别对于有交互信息的网站及进行电子商务活动的网站尤其重要。

站点涉及银行信用卡支付问题,用户资料信息保密问题等。Web 页面随时会传输这些重要信息,所以一定要确保安全性。一旦用户信息被黑客捕获泄露,客户在进行交易时,就不会有安全感,甚至后果严重。

1. 目录设置

Web 安全的第一步就是正确设置目录。目录安全是 Web 安全性测试中不可忽略的问题。如果 Web 程序或 Web 服务器的处理不当,通过简单的 URL 替换和推测,会将整个 Web 目录暴露给用户,这样会造成 Web 的安全性隐患。每个目录下应该有 index.html 或 main.html 页面,或者严格设置 Web 服务器的目录访问权限,这样就不会显示该目录下的所有内容,从而提高安全性。

2. SSL

很多站点使用 SSL(Security Socket Layer)安全协议进行传送。

SSL 表示安全套接字协议层,是由 Netscape 首先发表的网络数据安全传输协议。SSL 是利用公开密钥/私有密钥的加密技术,在位于 HTTP 层和 TCP 层之间,建立用户和服务器之间的加密通信,从而确保所传送信息的安全性。

任何用户都可以获得公共密钥来加密数据,但解密数据必须通过对应的私人密钥。SSL 是工作在公共密钥和私人密钥基础上的。

当用户进入到一个 SSL 站点是因为浏览器出现了警告消息,而且在地址栏中的 HTTP 变成 HTTPS。如果开发部门使用了 SSL,测试人员需要确定是否有相应的替代页面,适用于 3.0 以下版本的浏览器,这些浏览器不支持 SSL。当用户进入或离开安全站点的时候,请确认有相应的提示信息。做 SSL 测试时,需要确认是否有连接时间限制,超过限制时间后会出现什么情况等。

3. 登录

如图 9-5 所示,很多站点都需要用户先注册后登录使用,从而校验用户名和匹配的密码,以验证他们的身份,阻止非法用户登录。这样对用户是方便的,他们不需要每次都输入个人资料。

测试人员需要验证系统阻止非法的用户名/口令登录,而能够通过有效登录。主要的测试内容有:

- 测试用户名和输入密码是否有大小写区别;
- 测试有效和无效的用户名和密码;
- 测试用户登录是否有次数限制,是否限制从某些 IP 地址登录;
- 假设允许登录失败的次数为 3 次,那么在用户第三次登录的时候输入正确的用户名和口令,测试是否能通过验证;
- 测试口令选择是否有规则限制;

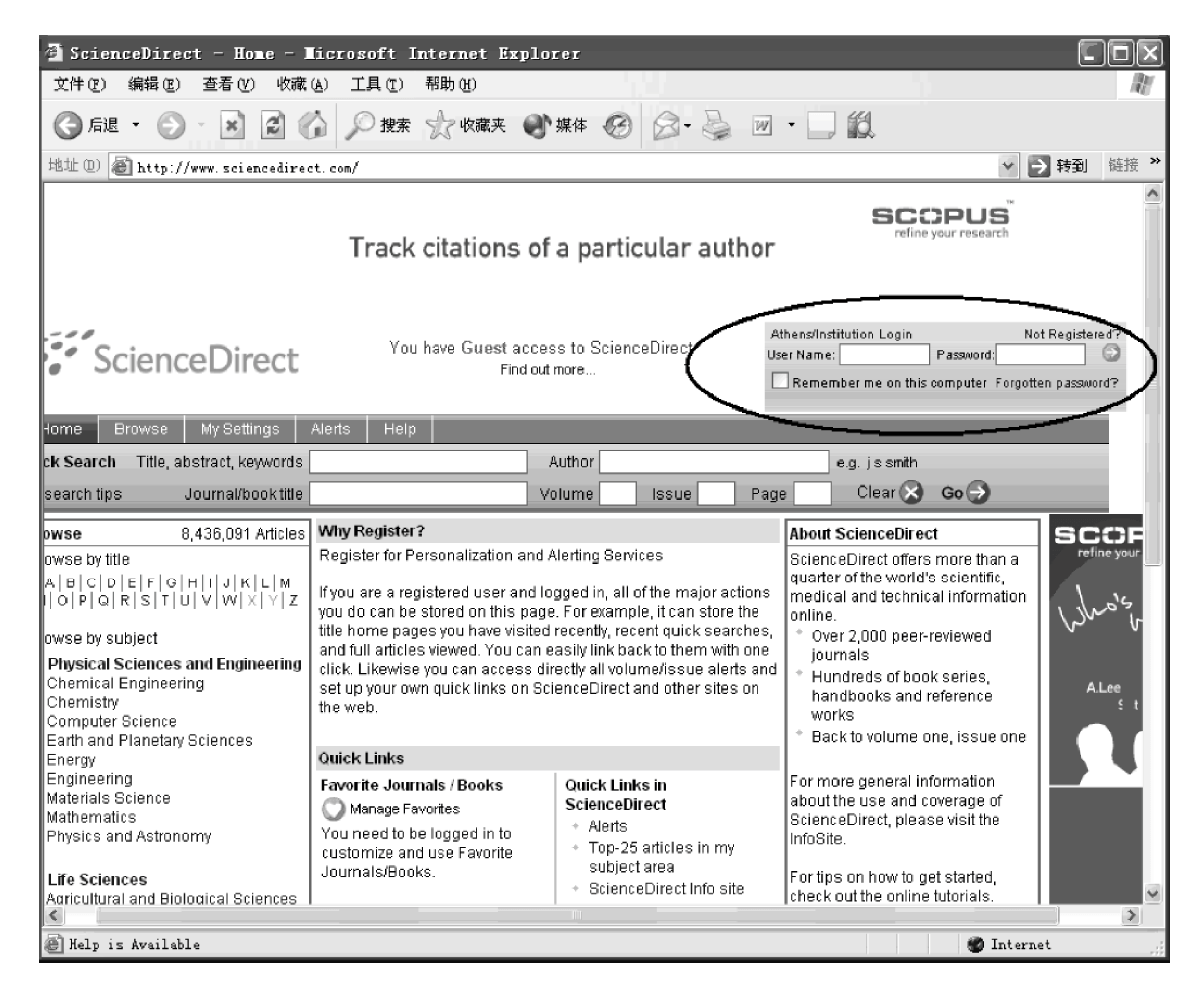


图 9-5　用户登录设置

- 测试哪些网页和文件需要登录才能访问和下载；
- 测试是否可以不登录而直接浏览某个页面；
- 测试 Web 应用系统是否有超时的限制，也就是说，用户登录后在一定时间内（例如 15 分钟）没有点击任何页面，是否需要重新登录才能正常使用。

另外，许多站点在登录邮箱时，也会有安全性提示。这里以 YAHOO 为例，如图 9-6 所示是点击 YAHOO 的信箱图标时弹出的对话框，提示用户网页通过安全链接。这样用户就会安心地登录邮箱了。

4. 日志文件

为了保证 Web 应用系统的安全性，日志文件是至关重要的。需要测试相关信息是否写进了日志文件、是否可追踪。

在后台，要注意验证服务器日志工作是否正常。对于日志文件主要的测试内容有：

- 日志是否记录所有的事务处理；



图 9-6　安全链接提示

- CPU 的占有率是否很高；
- 是否有例外的进程占用；
- 是否记录失败的注册企图；
- 是否记录被盗信用卡的使用；
- 是否在每次事务完成的时候都进行保存；
- 是否记录 IP 地址；
- 是否记录用户名等。

5. 脚本语言

脚本语言是常见的安全隐患。每种语言的细节有所不同。有些脚本允许访问根目录，其他脚本只允许访问邮件服务器。但是有经验的黑客可以利用这些缺陷，将服务器用户名和口令发送给他们自己，从而攻击和使用服务器系统。

测试人员需要找出站点使用了哪些脚本语言，并研究该语言的缺陷。

服务器端的脚本常常构成安全漏洞，这些漏洞又常常被黑客利用。所以，还需要检验没有经过授权，就不能在服务器端放置和编辑脚本的问题。最好的办法是订阅一个讨论站点使用的脚本语言安全性的新闻组。

6. 加密

当使用了安全套接字时，还要测试加密是否正确，检查信息的完整性。

9.5　可用性/可靠性测试

可用性/可靠性方面一般采用手工测试的方法进行评判，可用性测试的内容包括导航测试、Web 图形测试和图形用户界面测试等，可靠性测试的内容较直观。

9.5.1 导航测试

导航描述了用户在一个页面内操作的方式,在不同的用户接口控制之间,例如按钮、对话框、列表和窗口等,或在不同的连接页面之间。

导航测试的主要测试目的是检测一个 Web 应用系统是否易于导航,具体内容包括:

- 导航是否直观;
- Web 系统的主要部分是否可通过主页存取;
- Web 系统是否需要站点地图、搜索引擎或其他的导航帮助。

在一个页面上放太多的信息往往起到与预期相反的效果。Web 应用系统的用户大多是趋向于目的驱动,很快地扫描一个 Web 应用系统,看是否有满足自己需要的信息,如果没有,就会很快地离开。很少有用户愿意花时间去熟悉 Web 应用系统的结构,因此,Web 应用系统导航帮助要尽可能地准确。

导航的另一个重要方面是 Web 应用系统的页面结构、导航、菜单、连接的风格是否一致。确保用户凭直觉就知道 Web 应用系统里面是否还有内容,内容在什么地方。

Web 应用系统的层次一旦决定,就要着手测试用户导航功能,应该让最终用户参与这种测试,提高测试质量。

导航条测试用例如表 9-6 所示。

表 9-6 导航条测试用例示例

测试用例号	操作描述	数据	期望结果	实际结果
9.16	1. 执行一个搜索,至少搜索到 10 项相关商品信息 2. 以一件商品为单位向下滚动	查询=	搜索结果有 10 个或 10 个以上的相关商品信息;在没有到达搜索列表页面底部时,前面的商品列表滚动出屏幕,后面的商品不断从屏幕下方出现	一致/不一致
9.17	1. 执行一个搜索,至少搜索到 5 个页面的输出 2. 以页面为单位向下滚动	查询=	搜索结果有 5 个或 5 个以上的相关页面;在没有到达搜索列表的底部时,当前的屏幕内容向上滚动一屏,下一屏出现	一致/不一致

9.5.2 Web 图形测试

在 Web 应用系统中,适当的图片和动画既能起到广告宣传的作用,又能起到美化页面的功能。一个 Web 应用系统的图形可以包括图片、动画、边框、颜色、字体、背景、按钮等。图形测试的内容有:

- 要确保图形有明确的用途,图片或动画不要胡乱地堆在一起,以免浪费传输时间。

Web 应用系统的图片尺寸要尽量地小，并且要能清楚地说明某件事情，一般都链接到某个具体的页面。

- 验证所有页面字体的风格是否一致。
- 背景颜色应该与字体颜色和前景颜色相搭配。通常来说，使用少许或尽量不使用背景是个不错的选择。如果想使用背景，那么最好使用单色的，和导航条一起放在页面的左边。另外，图案和图片可能会转移用户的注意力。
- 图片的大小和质量也是一个很重要的因素，一般采用 JPG 或 GIF 压缩，最好能使图片的大小减小到 30KB 以下。
- 验证文字回绕是否正确。如果说明文字指向右边的图片，应该确保该图片出现在右边。不要因为使用图片而使窗口和段落排列古怪或者出现孤行。
- 根据图片能否正常加载，用来检测网页的输入性能好坏。如果网页中有太多图片或动画插件，就会导致传输和显示的数据量巨大、减慢网页的输入速度，有时会影响图片的加载。

如图 9-7 所示，网页无法载入图片时，就会在其显示位置上显示错误提示信息。

Web 图形测试用例如表 9-7 所示。

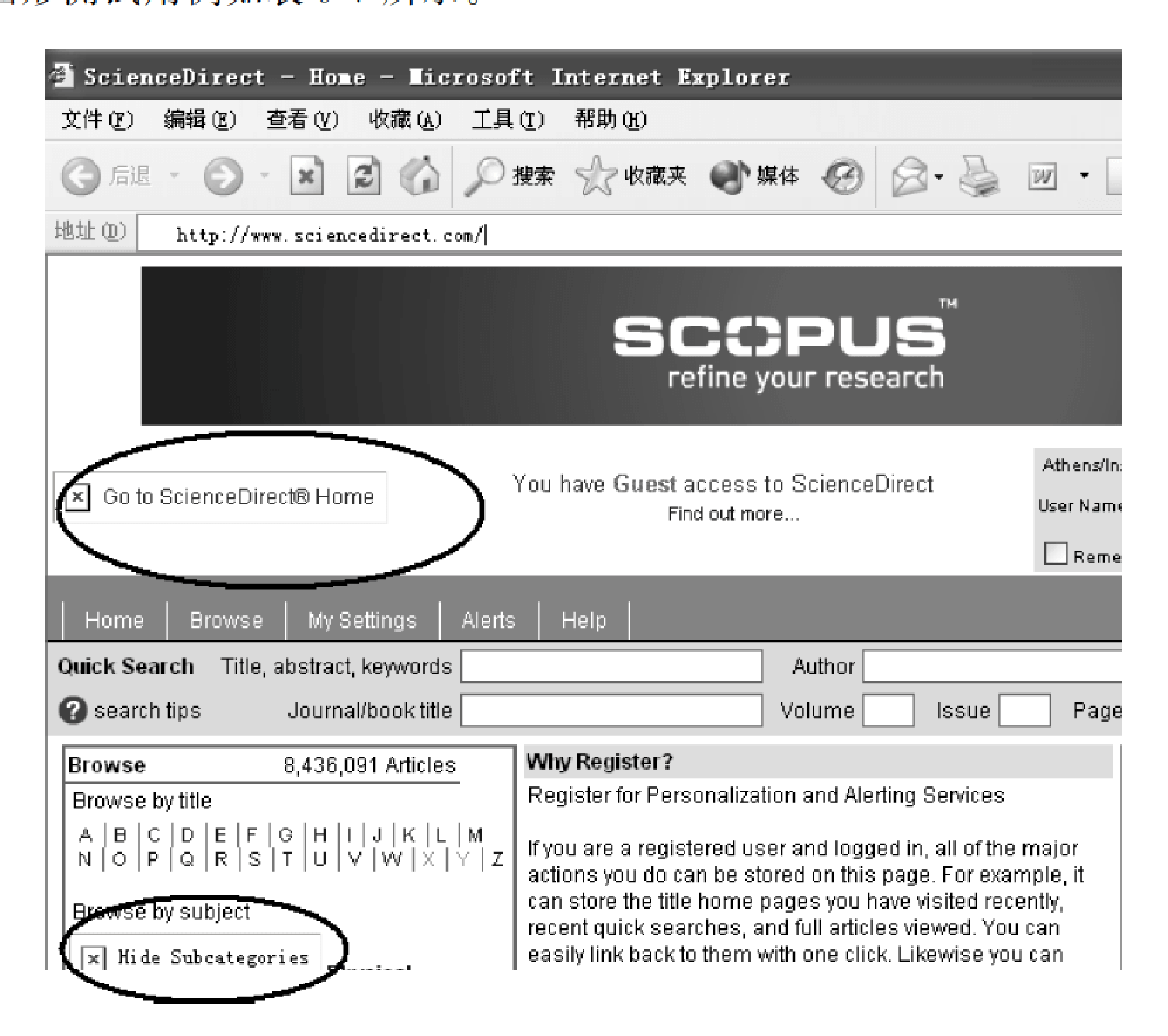


图 9-7　网页无法载入图片的提示信息

表 9-7 Web 图形测试用例示例

测试用例号	操作描述	数据	期望结果	实际结果
9.18	查看图形/图像	页面＝ 浏览器＝	在选择的浏览器中,图形/图像显示正确	一致/不一致

9.5.3 图形用户界面(GUI)测试

现在一般人都有使用浏览器浏览网页的经历,界面对不懂技术的用户来说非常重要,所以搞好界面测试也很关键。

(1) 整体界面测试

整体界面是指整个 Web 应用系统的页面结构设计,是给用户的一个整体感。例如,当用户浏览 Web 应用系统时是否感到舒适,是否凭直觉就知道要找的信息在什么地方,整个 Web 应用系统的设计风格是否一致等。

对整体界面的测试过程,其实是一个对最终用户进行调查的过程。一般 Web 应用系统采取在主页上做一个调查问卷的形式,来得到最终用户的反馈信息。因此测试需要外部人员参加,特别是终端用户的参与。

(2) 界面测试要素

界面测试要素主要包括:符合标准和规范,具有直观性、一致性、灵活性、舒适性、正确性、实用性。

① 直观性

直观性包含的问题有:

用户界面是否洁净、不奇怪、不拥挤,界面不应该为用户制造障碍。所需功能或者期待的响应应该明显,并在预期出现的地方。

- 界面组织和布局是否合理?
- 是否允许用户轻松地从一个功能转到另一个功能?
- 下一步做什么是否明显?
- 任何时刻都可以决定放弃或者退回、退出吗?
- 输入得到承认了吗?
- 菜单或者窗口是否深藏不露?
- 有多余功能吗?
- 软件整体抑或局部是否做得太多?
- 是否有太多特性把工作复杂化了?
- 是否感到信息太庞杂?
- 如果其他所有努力失败,帮助系统能否帮忙?

② 一致性

一致性包含的问题有：

- 快速键和菜单选项：在 Windows 中按 F1 键总是得到帮助信息？
- 术语和命令：整个软件使用同样的术语吗？特性命名一致吗？
- 软件是否一直面向同一级别用户？
- 按钮位置和等价的按键：大家是否注意到对话框有 OK 按钮和 Cancle 按钮时，OK 按钮总是在上方或者左方，而 Cancle 按钮总是在下方或右方。同样原因，Cancle 按钮的等价按键通常是 Esc 键，而选中按钮的等价按钮通常是与 Enter 键保持一致。

③ 灵活性

灵活性包含的问题有：

- 状态跳转：灵活的软件实现同一任务有多种选择方式。
- 状态终止和跳过，具有容错处理能力。
- 数据输入和输出：用户希望有多种方法输入数据和查看结果。例如，在写字板插入文字可用键盘输入、粘贴。

④ 舒适性

舒适性包含的问题有：

- 恰当：软件外观和感觉应该与所做的工作和使用者相符。
- 错误处理：程序应该在用户执行严重错误的操作之前提出警告，并允许用户恢复由于错误操作导致丢失的数据。如大家认为 undo/redo 是当然的。
- 性能：速度快不见得是好事。要让用户看清程序在做什么。

(3) 界面测试内容

用户界面测试主要包括以下几个方面的内容：

① 站点地图和导航条

测试站点地图和导航条位置是否合理、是否可以导航等。内容布局是否合理，滚动条等简介说明。

确认测试的站点是否有地图。有些网络高手可以直接去自己要去的地方，而不必打开许多页面。另外新用户在网站中可能会迷失方向。站点地图和/或导航条可以引导用户进行浏览。需要验证站点地图是否正确。确认地图上的链接是否确实存在。地图是否包括站点上的所有链接。

② 使用说明

说明文字是否合理，位置是否正确。

应该确认站点是否有使用说明。一般要确保站点具有使用说明，因为即使网站很简单，也可能有用户在某些方面需要证实一下。测试人员需要测试说明文档，验证说明是正

确的。还可以根据说明进行操作，确认出现预期的结果。

③ 背景/颜色

背景/颜色是否正确、美观，是否符合用户需求。

由于 Web 日益流行，很多人把它看作图形设计作品。不幸的是，有些开发人员对新的背景颜色更感兴趣，以至于忽略了这种背景颜色是否易于浏览。例如在紫色图片的背景上显示黄色的文本。这种页面显得“非常高贵”，但是看起来很费劲。通常来说，使用少许或尽量不使用背景是个不错的选择。如果想用背景，那么最好使用单色的，和导航条一起放在页面的左边。另外，图案和图片可能会转移用户的注意力。

④ 图片

无论作为屏幕的聚焦点或作为指引的小图标，一张图片都胜过千言万语。有时，告诉用户一个东西的最好办法就是将它展示给用户。但是，带宽对客户端或服务器来说都是非常宝贵的，所以要注意节约使用内存。

相关测试内容包括：

- 保证图片有明确用途：是否所有的图片对所在的页面都是有价值的，或者它们只是浪费带宽；
- 图片的大小和质量：图片是否使用了 *.GIF、*.JPG 的文件格式。是否能使图片的大小减小到 30KB 以下；
- 所有图片能否正确载入和显示：通常，不要将大图片放在首页上，因为这样可能会使用户放弃下载首页。如果用户可以很快看到首页，他可能会浏览站点，否则可能放弃；
- 背景颜色是否和字体颜色以及前景颜色搭配。

⑤ 表格

表格测试的相关内容：

- 需要验证表格是否设置正确？
- 用户是否需要向右滚动页面才能看见产品的价格？
- 把价格放在左边，而把产品细节放在右边是否更有效？
- 每一栏的宽度是否足够宽，表格里的文字是否都有折行？
- 是否有因为某一格的内容太多，而将整行的内容拉长？

表格测试用例示例如表 9-8 所示。

⑥ 回绕

需要验证的是文字回绕是否正确。如果说明文字指向右边的图片，应该确保图片出现在右边。不要因为使用图片而使窗口和段落排列古怪或者出现孤行。

另外，测试内容还包括测试页面在窗口中的显示是否正确、美观(在调整浏览器窗口大小时，屏幕刷新是否正确)。表单样式、大小和格式是否对提交数据进行验证(如果在页

面部分进行验证)等。链接的形式、位置是否易于理解等。

表 9-8　表格测试用例示例

测试用例号	操作描述	数　据	期 望 结 果	实际结果
9.19	查看表格	表格= 浏览器=	在选择的浏览器中,表格显示正确	一致/不一致

9.5.4　可靠性测试

可靠性测试很容易理解,如表 9-9 所示,直接给出可靠性测试示例。

表 9-9　可靠性测试用例示例

测试用例号	操 作 描 述	数　据	期 望 结 果	实际结果
9.20	在网站购物的同时,打印当前页面	商品=	商品能够成功购买,选择的页面也能打印成功,系统速度正常、性能稳定	一致/不一致
9.21	利用自动测试工具,每一分钟购买一次商品	商品 1= 商品 2= 商品 3= 商品 4= 商品 5= ⋮ 商品 n=	每件商品都能成功购买,系统速度正常、性能稳定	一致/不一致
9.22	5 个用户一起登录网站,并同时购买同一个商品	用户 1= 用户 2= 用户 3= 用户 4= 用户 5=	5 个用户都能在同一时间将相同的商品放在各自的购物车中	一致/不一致

9.6　配置和兼容性测试

需要验证应用程序可以在用户使用的机器上运行。如果用户是全球范围的,需要测试各种操作系统、浏览器、视频设置和 Modem 的速度。最后,还要尝试各种设置的组合。

1. 平台测试

市场上有很多不同的操作系统类型,最常见的有 Windows、UNIX、Linux 等。Web

应用系统的最终用户究竟使用哪一种操作系统，取决于用户系统的配置。这样，就可能会发生兼容性问题，同一个应用可能在某些操作系统下能正常运行，但在另外的操作系统下可能会运行失败。

因此，在 Web 系统发布之前，需要在各种操作系统下对 Web 系统进行兼容性测试。

2. 浏览器测试

浏览器是 Web 客户端核心的构件，需要测试站点能否使用 Netscape、Internet Explorer 或 Lynx 进行浏览。来自不同厂商的浏览器对 Java、JavaScript、ActiveX 或不同的 HTML 规格有不同的支持。并且有些 HTML 命令或脚本只能在某些特定的浏览器上运行。

例如，ActiveX 是 Microsoft 的产品，是为 Internet Explorer 而设计的，JavaScript 是 Netscape 的产品，Java 是 Sun 的产品等。另外，框架和层次结构风格在不同的浏览器中也有不同的显示，甚至根本不显示。不同的浏览器对安全性和 Java 的设置也不一样。

测试浏览器兼容性的一个方法是创建一个兼容性矩阵。在这个矩阵中，测试不同厂商、不同版本的浏览器对某些构件和设置的适应性。

大多数 Web 浏览器允许大量自定义。如图 9-8 所示，可以在选择安全性选项、选择文字标签的处理方式、选择是否启用插件等。不同的选择项对于网站的运行有各自不同的影响，因此测试时每个选项都要考虑。

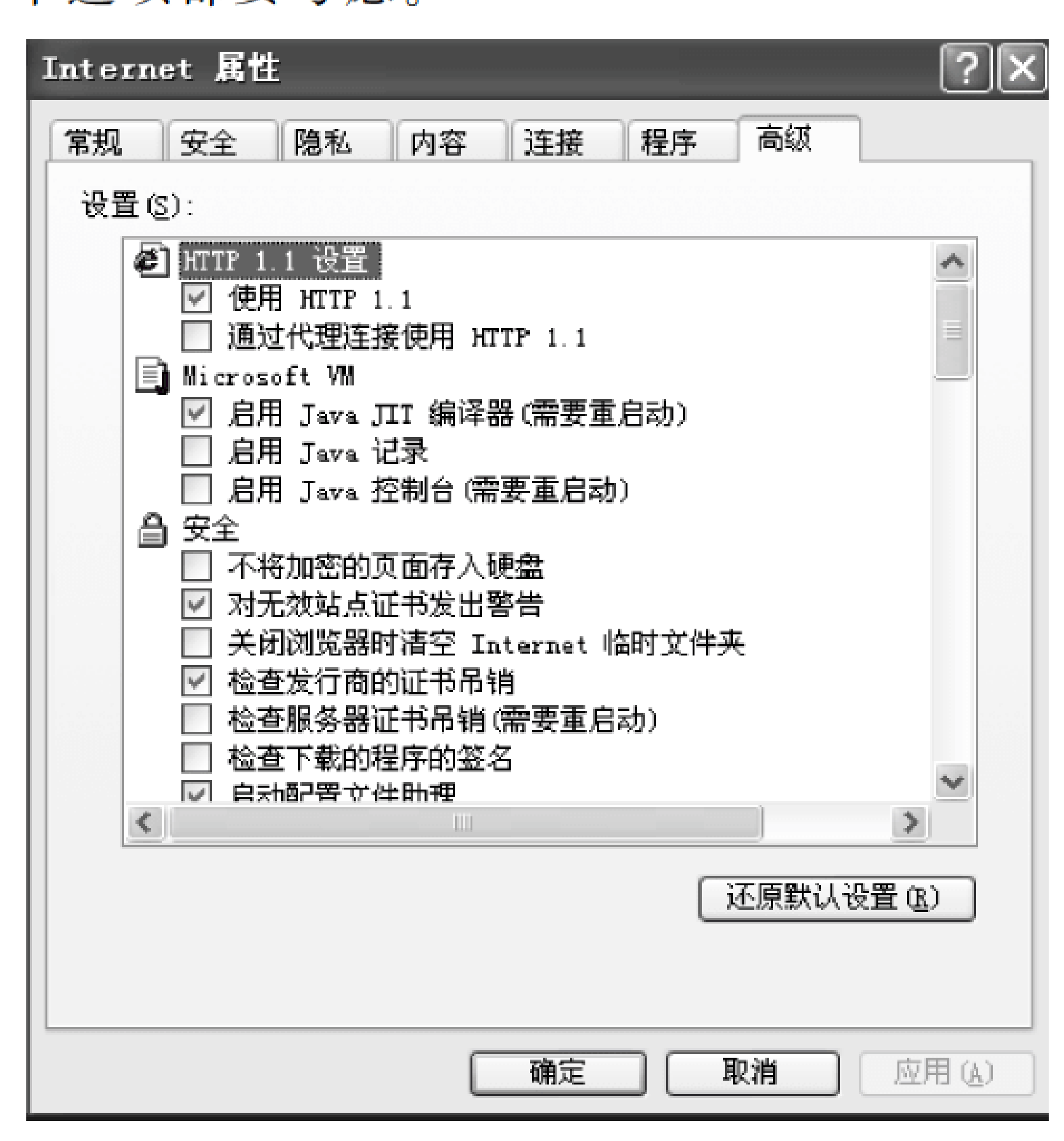


图 9-8 Internet Explorer 浏览器的可配置性

浏览器环境和测试平台的兼容性如表 9-10 所示。在不同的平台和浏览器组合中执行相同的测试用例,在执行后核对结果并将兼容性填入表格中。

表 9-10 浏览器兼容表

浏览器 平台	Netscape Communicatior 4.5	Netscape Communicatior 4.7	Internet Explorer 4.01	Internet Explorer 5.0
Windows 98				
Windows 2000				
Windows NT				
Windows XP				
Windows NE				
Linux		—	—	
UNIX				
iMac				
Mac OS X				

3. 打印机测试

用户可能会将网页打印下来。因此网页在设计的时候要考虑到打印问题,注意节约纸张和油墨。有不少用户喜欢阅读而不是盯着屏幕,因此需要验证网页打印是否正常。有时在屏幕上显示的图片和文本的对齐方式可能与打印出来的东西不一样。测试人员至少需要验证订单确认页面打印是正常的。

4. 组合测试

最后需要进行组合测试。600×800 的分辨率在 MAC 机上可能不错,但是在 IBM 兼容机上却很难看。在 IBM 机器上使用 Netscape 能正常显示,但却无法使用 Lynx 来浏览。

如果是内部使用的 Web 站点,测试可能会轻松一些。如果公司指定使用某个类型的浏览器,那么只需在该浏览器上进行测试。如果所有的人都使用 T1 专线,可能不需要测试下载施加。(但需要注意的是,可能会有员工从家里拨号进入系统)有些内部应用程序,开发部门可能在系统需求中声明不支持某些系统而只支持那些已设置的系统。但是,理想的情况是,系统能在所有机器上运行,这样就不会限制将来的发展和变动。

可以根据实际情况,采取等价划分的方法,列出兼容性矩阵。

5. 兼容性测试

兼容性测试用例如表 9-11 所示。

表 9-11 兼容性测试用例示例

测试用例号	操作描述	数据	期望结果	实际结果
9.23	将网站加入到收藏夹中,会话结束时再次调用	Web 页面=	网站正常打开和运行	一致/不一致
9.24	打开某个站点的多个会话	Web 页面=	每个会话都是可用的	一致/不一致
9.25	使用浏览器的打印功能	Web 页面=	选择的页面能够正常打印	一致/不一致
9.26	创建一个 Web 页面的快捷键,在结束会话后点击该快捷键	Web 页面=	网站正常打开和运行	一致/不一致

9.7 数据库测试

在 Web 应用技术中,数据库具有非常重要的作用,数据库为 Web 应用系统的管理、运行、查询和实现用户对数据存储的请求等提供空间。在 Web 应用中,最常用的数据库类型是关系型数据库,可以使用 SQL 对信息进行处理。

数据库测试是 Web 网站测试的一个基本组成部分。网站把相关的数据和信息存储在数据库中,从而提高搜索效率。很多站点把用户的输入数据也存放在数据库中。

对于测试人员,要真正了解后台数据库的内部结构和设计概念,制定详细的数据库测试计划,至少能在程序的某个流程点上并发地查询数据库。

1. 数据库测试的主要因素

数据库测试的主要因素有:数据完整性、数据有效性和数据操作和更新。

- 数据的完整性:测试的重点是检测数据损坏程度。开始时,损坏的数据很少,但随着时间的推移和数据处理次数的增多,问题会越来越严重。设定适当的检查点可以减轻数据损坏的程度。比如,检查事务日志以便及时掌握数据库的变化情况。
- 数据有效性:数据有效性能确保信息的正确性,使得前台用户和数据库之间传送的数据是准确的。在工作流上的变化点上检测数据库,跟踪变化的数据库,判断其正确性。
- 数据操作和更新:根据数据库的特性,数据库管理员可以对数据进行各种不受限制的管理操作。具体包括:增加记录、删除记录、更新某些特定的字段。

2. 数据库测试的相关问题

除了上面的数据库测试因素,测试人员需要了解的相关问题有:

- 数据库的设计概念；
- 数据库的风险评估；
- 了解设计中的安全控制机制；
- 了解哪些特定用户对数据库有访问权限；
- 了解数据的维护更新和升级过程；
- 当多个用户同时访问数据库处理同一个问题，或者并发查询时，确保可操作性；
- 确保数据库操作能够有足够的空间处理全部数据，当超出空间和内存容量时能够启动系统扩展部分。

围绕上述的测试因素和测试的相关问题，就可以设计具体的数据库测试用例了。

3. 测试用例

在学校的网站上，成绩查询系统是一个常见的 Web 程序。学生可以通过浏览器页面访问 Web 服务器，Web 服务器再从数据库服务器上读取数据。

如表 9-12 所示是一个学生基础课成绩表的结构示例。这里定义了表的各项字段名、字段类型及其含义。

表 9-12　学生成绩表的结构示例

字段名	字段类型	含　义	注释
S_No	整型	学号	非空
S_Name	字符串类型	学生姓名	非空
S_Dep	字符串类型	所在系	
S_Class	字符串类型	所在班级	
M_Score	数值型	数学成绩	
E_Score	数值型	英语成绩	
C_Score	数值型	计算机成绩	

如表 9-13 所示是对应的数据库测试用例示例。实际测试结果和期望结果是否一致要取决于数据库的性能高低。

表 9-13　数据库测试用例示例

测试用例号	操作描述	数　据	期望结果	实际结果
9.27	指定学号来查询成绩	S_No=	输出该学号对应学生的所有成绩情况	一致/不一致
9.28	指定一个有效且不重名的学生姓名来查询成绩	S_Name=	输出该学生的所有成绩情况	一致/不一致
9.29	指定一个有效且重名的学生姓名来查询成绩	S_Name=	输出该学生的所有成绩情况	一致/不一致

续表

测试用例号	操作描述	数　据	期望结果	实际结果
9.30	指定一个不存在的学生姓名来查询成绩	S_Name=	学生记录没有找到，建议重新输入	一致/不一致
9.31	指定一个有效的学生姓名和所在班级组合条件来查询该学生的相关成绩	S_Name= S_Class=	输出该学生的所有成绩情况	一致/不一致
9.32	指定一个有效的学生姓名和一个不存在的班级来查询该学生的相关成绩	S_Name= S_Class=	系别没有找到，但列出该学生的所有成绩情况	一致/不一致
9.33	指定一个有效的学生姓名和所在系别组合条件来查询该学生的相关成绩	S_Name= S_Dep=	输出该学生的所有成绩情况	一致/不一致
9.34	指定一个有效的学生姓名和一个不存在的系别来查询该学生的相关成绩	S_Name= S_Dep=	系别没有找到，但列出该学生的所有成绩情况	一致/不一致
9.35	1. 根据学号查询到该学生的英语成绩 2. 数据库管理员更改该学生的英语成绩 3. 根据学号再次查询该学生的英语成绩	S_No= E_Score=	1. 输出该学生英语成绩 2. 更新数据库 3. 给出该学生更新后的英语成绩	一致/不一致
9.36	并发执行以下操作： 1. 数据库管理员增加一名新同学的记录 2. 用户查询这名新同学的相关信息	S_No= S_Name= S_Dep= S_Class= M_Score= E_Score= C_Score=	查询结果可能给出不完整的相关信息，比如有空的字段	一致/不一致
9.37	N个用户同时执行相同的查询操作	(要查询的)字段名= 用户数=	在可以接受的响应时间内，所有用户得到正确的显示结果	一致/不一致

小　　结

本章介绍了 Web 网站测试的几个方面和相关的测试技术。

Web 测试相对于非 Web 测试来说都是更具挑战性的工作。用户对 Web 页面质量

有很高的期望。

功能测试是检测网站功能的正确性，其中包括页面内容测试、链接测试、表单测试、Cookies 测试和设计语言测试等。

性能测试确保网站服务器在规定的参数内响应浏览器的请求。作为性能测试的一部分，负载测试评估网站满足负载要求的能力。负载测试评估系统在处理大量用户的并发要求时的功能如何。压力测试是使系统能满足不同的负载。连接速度测试则是对打开网页的响应速度测试。

安全测试是为了确保重要和机密信息的安全性，试图找到应用程序的安全缺陷。

可用性测试是指通过观察用户与站点的交互，评估一个站点是否用户友好。其中导航测试是指通过访问页面、图像、链接及其他页面组件，确保用户可以完成希望的任务。

配置和兼容性测试保证了应用程序在各种硬件和软件环境下的功能都是正确的。

数据库测试检查存储数据的完整性，而存储数据通常是指网站使用的产品信息。

习　　题

1. 简述 Web 网站的测试内容。
2. 功能测试包括哪些方面？
3. 负载/压力测试的作用是什么？
4. 概括安全性测试中的登录测试内容。
5. 简述兼容性测试。

第 10 章

软件测试职业

本章概述

本章阐述了软件测试职业和职位。介绍了获取软件测试资源的途径并对软件测试工程师的素质提出了要求。

10.1 软件测试职业和职位

软件测试工作随着软件产品的开发所起的作用越来越重要,这是软件行业二十几年的实践所证明的一个道理。

以微软公司为例,微软以前的产品时时会发生崩溃,死机等现象,而今天的产品相比5年前的产品,不仅功能要强大得多,稳定性也要好得多。这是因为微软公司重视测试工作,测试人员越来越多,如今微软的软件测试人员是开发人员的1.5～2.5倍。其次,测试人员越来越有经验,测试工作也就越做越好。正是由于清晰地认识到了软件测试的重要性,微软的产品质量才有了明显的提高。

最初,微软公司认为测试不重要,重要的是开发人员。通常一个团队中有几百个开发人员,但只有几个测试人员,并且开发人员的待遇要比测试人员的待遇高得多。经过多年的实践后,微软公司发现,去修正那些出现问题的产品所花的钱,比多聘用几个测试人员的费用要多得多,所以,开始不断地聘用测试人员。

在当前中国的软件业迅速发展,并且在向国际软件产业市场迈进的进程中,在软件开发过程越来越工程化的规范下,对软件测试的重视程度也空前提高和强化,软件开发过程对专业测试人员的需求也不断地增加。目前国内专业化的软件测试人员无论从数量上还是质量上,与国外同行业相比均明显不足。

这样就给从事软件测试工作带来了许多就业机会。当前软件测试技术的职业市场表明,具有一定测试经验的软件测试工程师很受市场青睐,供不应求。目前,软件测试工作

越来越得到足够重视，现在测试人员的待遇和开发人员的待遇非常接近。

软件开发时，首先要组建一个开发团队，要决定这个团队应该有多少人参加，需要什么技术的人员参加，项目经理是谁，多少个开发人员？多少个测试人员？多少个程序经理？这些问题搞清楚了，开发团队就基本建立起来了。

10.1.1　测试团队的基本构成

测试团队的构成，从理论上说，和其规模没有多大关系，也就是人们常说的"麻雀虽小，五脏俱全"。如果项目很小，测试小组就一个人，那么这个人就要扮演不同的角色。一般来看，一个比较健全的测试部门应该具有下面这些角色。

- 测试经理：负责人员招聘、培训、管理、资源调配、测试方法改进等。
- 实验室管理人员：负责设置、配置和维护实验室的测试环境，主要是负责服务器和网络环境等。
- 内审员：负责审查流程，并提出改进流程的建议；建立测试文档所需的各种模板，检查软件缺陷描述及其他测试报告的质量等。
- 测试组长：是相关的业务专家，负责项目的管理、测试计划的制订、项目文档的审查、测试用例的设计和审查、任务的安排，负责和项目经理、开发组长的沟通等。
- 一般(初级)测试工程师：执行测试用例和相关的测试任务。

对于比较大规模的测试团队，测试工程师分为三个层次：初级测试工程师、测试工程师、资深(高级)测试工程师等，同时还设立自动化测试工程师、系统测试工程师和架构工程师。

对于规模很小的测试小组，可能没有设置测试经理，只有测试组长，这时测试组长承担测试经理的部分责任，如参加面试工作、资源管理、团队发展等，并且要做内审员的工作，检查软件缺陷描述及其他测试报告的质量等。资深测试工程师不仅要负责设计规格说明书的审查、测试用例的设计等，还要设置测试环境，即承担实验室管理人员的责任。

10.1.2　测试人员职位及其责任

在上面介绍了团队的基本构成，为了更好地理解团队中的每位成员所起的作用，就需要清楚不同的角色所应该承担的责任。

主要角色的责任，先从一般(初级)测试工程师开始，再介绍资深测试工程师，最后到测试经理。这个过程有利于读者理解他们的责任，测试工程师虽然和初级测试工程师责任不一样，但可以肯定的是测试工程师能做好所有要求初级测试工程师做好的工作。

不同层次的测试工程师责任有一定的区别，但都是技术工作，主要任务是设计和执行各种测试任务，是测试工作的基础。

下面对软件测试各职位及其责任做详细的介绍。

(1) 初级测试工程师

初级测试工程师的责任比较简单,还不具备完全独立的工作能力,需要测试工程师或资深测试工程师的指导,要求比较低,主要有下列7项责任。

- 了解和熟悉产品的功能、特性等;
- 验证产品在功能、界面上是否和产品规格说明书一致;
- 按照要求,执行测试用例,进行功能测试、验收测试等,并能发现所暴露的问题;
- 清楚地描述所出现的软件问题;
- 努力学习新技术和软件工程方法,不断提高自己的专业水平;
- 使用简单的测试工具;
- 接受测试工程师的指导,执行主管所交代的其他工作。

(2) 测试工程师

测试工程师的责任相对多些,熟悉测试流程、测试方法和技术,参与自动化测试,具有独立的工作能力,但基本上以执行测试为主,主要责任如下:

- 熟悉产品的功能、特性,审查产品规格说明书;
- 根据需求文档或设计文档,可以设计功能方面的测试用例;
- 根据测试用例,执行各种测试,发现所暴露的问题;
- 全面使用测试工具,包括测试脚本的编写;
- 安装、设置简单的系统测试环境;
- 报告所发现的软件缺陷,审查软件缺陷,跟踪缺陷修改的情况,直到缺陷关闭;
- 写测试报告;
- 负责对初级测试工程师的指导,执行主管所交代的其他工作。

(3) 资深测试工程师

资深测试工程师不仅具有良好的技术、产品分析能力、解决问题能力、丰富的测试工作经验,而且有较好的编程、自动化测试经验,熟悉测试流程、测试方法和技术,解决测试经理工作中可能遇到的各种技术问题。主要责任如下:

- 负责系统一个或多个模块的测试工作;
- 制订某个模块或某个阶段的测试计划、测试策略;
- 设计测试环境所需的系统或网络结构,安装、设置复杂的系统测试环境;
- 熟悉产品的功能、特性,审查产品规格说明书,并提出改进要求;
- 审查代码;
- 验证产品是否满足了规格说明书所描述的需求;
- 根据需求文档或设计文档,设计复杂的测试用例;
- 负责对测试工程师的指导,执行主管所交代的其他工作。

（4）测试实验室管理员

测试实验室管理员主要负责建立、设置和维护测试环境，保证测试环境的稳定运行，其主要责任如下：

- 负责测试环境所需的网络规划和建设，维护网络的正常运行；
- 建立、设置和维护测试环境所需的应用服务器和软件平台；
- 申请所需要的新的硬件资源、软件资源，协助有关部门进行采购、验收；
- 对使用实验室的硬件、软件资源的权限进行设计、设置，保证其安全性；
- 安装新的测试平台、被测试的系统等；
- 优化测试环境，提高测试环境中网络、服务器和其他设备运行的性能。

（5）软件包构建或发布工程师

发布工程师在 QA 工作中起着很重要的角色，负责测试产品的上载、打包和发布，其主要责任是：

- 负责源程序代码管理系统的建立、管理和维护；
- 保证文件名定义规范，建立合理的程序文件结构和存储目录结构；
- 为程序的编译、连接等软件包构造建立自动处理文件；
- 保证测试最新的产品包上传到相应的服务器上，并确认各模块或组件之间相互匹配；
- 每天为各项目新的或修改的代码重新构造新的软件包。确保不含病毒，不缺图片和各种文件；
- 负责软件包的接收、发送、存储和备份等。

（6）测试组长

测试组长一般具备资深测试工程师的能力和经验，可能在技术上相对弱些，不是小组内最强的，其责任偏重测试项目的计划、跟踪和管理，同时负责测试小组的团队的管理和发展。其主要责任如下：

- 负责测试小组的管理或参与测试团队的管理；
- 负责一个独立的测试项目；
- 制订整个项目的测试计划、测试策略，包括风险评估、日程表安排等；
- 熟悉产品的功能、特性，审查产品规格说明书，并提出改进意见；
- 审查系统、程序设计说明书；
- 验证产品是否满足了规格说明书所描述的需求；
- 实施软件测试，并对软件问题进行跟踪分析和报告，推动测试中发现的问题及时合理地解决；
- 编写项目的整体测试报告，保证产品质量；
- 对竞争者的产品进行分析，提出对软件的进一步改进的要求，并且评估改进方案

是否合理；

- 负责测试项目内部的资源、任务安排；
- 监督测试流程的执行，并将执行过程中所发现的问题反馈给测试经理或项目经理；
- 为团队成员提供技术指导，协助主管或测试经理工作。

（7）测试经理

测试经理的主要工作在团队、资源和项目等管理上，不同于测试组长。测试组长主要集中在项目管理上，一般不负责测试人员的招聘、流程定义等管理工作，而且偏重技术。测试经理对产品的质量负全面责任，有责任向公司最高管理层反映软件开发过程中管理问题或产品中的质量问题，使公司能全面掌握生产和质量状况。其主要责任如下：

- 负责整个测试团队或部门的管理，包括测试岗位责任的定义，组织团队结构的建立和优化，团队的建设和发展，培训活动的组织，员工的激励等；
- 负责一个完整产品的软件测试和质量保证等工作，包括项目组长的指定、项目资源的安排、项目进度的跟踪、项目审查和总结等；
- 负责测试部门年度/季度计划及预算的编写、实施和评估；
- 促进质量文化的普及，使整个开发团队的每位成员都有一个正确的客户和质量观念；
- 负责测试人员的招聘、考核等方面的工作，协助人力资源部门；
- 定义、实施软件测试流程或整个开发周期流程，并收集、处理流程实施中所存在问题，最终不断改进流程；
- 审查项目的测试计划、测试策略等，包括资源调度和平衡、风险评估等；
- 和其他部门协调，参加多方会议解决产品规格说明、设计等问题；
- 审查系统、程序设计说明书；
- 实施软件测试，并对软件问题进行跟踪分析和报告，推动测试中发现的问题能及时合理地解决；
- 审查项目的测试报告，进行产品质量的分析，提交质量分析报告；
- 对竞争者的产品进行分析，提出对软件进一步改进的要求并且评估改进方案是否合理。

10.2 软件测试资源的获取途径

软件测试工作随着软件产品开发的规范化、工程化已经越来越得到重视。要使得软件产品质量得到保证，并能在市场竞争中获胜，软件测试就显得尤为重要。要获得快捷可靠的软件测试资源，一般可以从正规的培训会议，相关的网站及从事软件测试的专业组织这三种途径中获取。

10.2.1　正规的培训会议

一般获取软件测试资源是参加正规的培训会议。美国及英、法等国通常每年都会在召开类似的国际会议，这些会议为软件测试人员提供了良好的获取资源的机会，会议资料包括最基础的软件测试内容及技术含量极高的新技术。它提供了软件测试的同行进行面对面的交流，讨论解决的相关方案及策略。一般比较规范的会议有：

（1）国际软件测试会议

由美国质量保证学会主办，由来自软件测试和质量保证行业的专家进行讲授。

（2）软件测试分析和评审

由软件质量工程学会主办，会议讨论的焦点主要集中在软件测试和软件工程方面。

（3）软件质量国际会议

由美国质量协会软件分会主办，也是提供从事软件测试和质量保证人士交流的机会。

（4）软件测试国际会议

由软件质量系统主办，是关于软件测试方面的演示、指导、讨论和经验交流的会议。

（5）软件质量世界会议

由国际软件质量协会主办，它是由学术及产业两个方面的顶尖专家组成，交流软件质量、软件过程改进及软件开发等方面的见解及经验。

10.2.2　相关的网络

因特网上拥有关于软件测试的丰富信息。虽然搜索“software testing”或者“software test”总可以找到一些资料，但是下列网站可以作为入门向导：

（1）Bug Net(www.bugnet.com)

公布在商业软件中发现的软件缺陷，并指出相应的修复措施。

（2）Software Testing Hotlist(www.io.com/～wazmo/qa)

列出了许多与软件测试相关的网站和文章的链接。

（3）Software Testing Online Resources(www.mtsu.edu/～storm)

自称一系列软件测试联机资源……旨在成为软件测试研究者和从业者的门户网站。

（4）QA Forums(www.qaforums.com)

提供软件测试，自动化测试、测试管理、测试工具等主题的即时讨论。

（5）comp.risks 新闻组描述和分析近期的软件失败。

10.2.3　从事软件测试的专业组织

从事软测试和软件质量保证的一些非商业性的组织，也是获取软件测试资源的一种途径，他们的网站提供了其专业范围的详细信息。

（1）美国软件测试协会

美国软件测试协会是个非盈利性专业服务组织，专注于软件测试的理解和实践的推

动。他们的文档提供了软件测试的丰富信息——强调实践而不是理论。

(2) 美国质量委员会

发表质量方面的刊物和文章,并管理认证质量工程师和认证软件质量工程师的任命。

(3) 美国计算机协会

已拥有教育和科学计算方面的 80000 多个会员。

(4) 软件质量委员会

其目标是成为那些志在把提高“质量”作为软件通用目标的人们的协会。

10.3 软件测试工程师的素质要求

软件测试是一项复杂而艰巨的任务,软件测试工程师的目标是尽早发现软件缺陷,以便降低修复成本。软件测试员是客户的眼睛,是最早看到并使用软件的人,所以应当站在客户的角度,代表应用客户说话,及时发现问题,力求使软件功能趋于完善。

很多比较成熟的软件公司都把软件测试视为高级技术职位。软件测试员的工作与程序员的工作对软件开发所起的作用是相当的。虽然软件测试员不一定是一个优秀的程序员,但是作为一个出色的软件测试员应当具备丰富的编程知识,掌握软件编程的基础内容,了解软件编程的过程,这无疑对出色完成软件测试任务具有很大的帮助。

通常软件测试工程师应具备如下素质:

(1) 具有较强的沟通能力

优秀的测试工程师必须能够同测试涉及的所有人进行沟通,具有与技术和非技术人员的交流能力。既要可以和用户交流,又要能同开发人员沟通,不幸的是这两类人没有共同语言。和用户交流的重点必须放在系统可以正确地处理什么和不可以处理什么上,尽量不使用专业术语。而和开发者沟通时,尽量要使用专业术语,这对用户反馈的相同信息,测试人员必须重新组织,以另一种方式表达出来,测试小组的成员必须能够同等地同用户和开发者沟通。

(2) 掌握比较全面的技术

就总体而言,开发人员对那些不懂技术的人持一种轻视的态度。一旦测试小组的某个成员做出了一个比较明显的错误断定,可能会被夸张地到处传扬,那么测试小组的可信度就会受到影响,其他正确的测试结果也会受到质疑。再者,由于软件错误通常依赖于技术,或者至少受构造系统所使用的技术的影响,所以测试人员掌握编程语言、系统构架、操作系统的特性、网络、表示层、数据库的功能和操作等知识,应该了解系统是怎样构成的,明白被测试软件系统的概念、技术,要建立测试环境、编写测试脚本,又要会使用软件工程工具。要做到这些,需要有几年以上的编程经验及对技术和应用领域的深刻理解。

（3）做优秀的外交家

优秀的测试人员必须能够同测试涉及的所有人进行良好的沟通。机智老练的外交手法有助于维护与开发人员的协作关系，幽默感同样也是很有帮助的。

测试人员应该把精力集中在查找错误上面，而不是放在找出是开发小组中哪个成员引入的错误。这样可以保证测试的否定性结果只是针对产品，而不是针对编程人员。也就是说要使用一种公正和公平的方式指出具体错误，这对于测试工作是有益的。一般来说，武断地对产品进行攻击是错误的。在遇到狡辩的情况下，一个幽默的批评将是很有帮助的。

（4）具有挑战精神

优秀的测试工程师在开发测试用例时使用的方法，与勘探专家在一个山洞中摸索前进的方法相类似。虽然周围可能存在大量死路，但是测试工程师要具有挑战性，这样才会促使他们向山洞中的深处探索，向一切没有去过的地方前进，最终可能会有一个大发现。

（5）具有准确的判断力

一个好的测试工程师具有一种先天的敏感性及准确的判断力，并且还能尝试着通过一些巧妙的变化去发现问题。同时，还具有强烈的质量追求，对细节的关注能力。测试人员要有高风险区的判断力以便将有限的测试针对重点环节来实现。

（6）做故障排除家

可以想象，开发人员会尽他们最大的努力将所有的错误解释过去，测试人员必须听每个人的说明，但必须保持高度警惕，才能做出决定。测试人员应该具有自我督促能力，才能保证每天的工作都能高质量完成，要善于发现问题的症结并及时清除。

（7）要有充分的自信心和耐心

开发人员指责测试人员出了错是常有的事，测试工程师必须对自己的观点有足够的自信心，对自己发现的缺陷有信心。如果没有信心或受开发人员影响过大，测试工作就缺乏独立性，程序中的漏洞或缺陷容易被忽略过去，就谈不上保证软件产品的质量。

还有一种情况也是常见的，软件产品设计规格说明书总是或多或少存在一些逻辑问题，编程人员和测试人员对那些有问题的功能存在争议，这时候信心会帮助测试人员发现产品设计中的问题。

有些软件测试工作需要难以置信的耐心。有时需要花费惊人的时间去分离、识别一个错误，需要对其中一个测试用例运行几十遍，甚至几百遍，了解错误在什么情况或什么平台下才发生。测试人员需要保持平静，尤其是在集中注意力解决困难问题的时候，特别是在测试执行阶段，面对成百上千个测试用例，要一个个去执行，还要在不同的测试环境上重复，耐心是必要的。当然，我们尽量使用测试工具去完成那些重复性的任务。

小　　结

本章介绍了软件测试职业的相关内容,阐述了软件测试职业和职位及软件测试资源的获取途径,以及对软件测试工程师应具备的基本素质要求。软件测试是一项批判性的工作,随着当今软件规模和复杂性的日益增加,进行专业化、高效的软件测试的要求也越来越高,从事软件测试人员的数量和质量都在提高。只有明确了软件测试工程师的目标及应具备的素质,才能做一个优秀的软件测试人员。同时,也为从事软件工作的人员提供了一个新的职位发展机会。

习　　题

1. 简述软件测试资源的获取途径。
2. 简述软件测试工程师应具备的素质。
3. 软件测试员的目标是什么?
4. 谈谈你对今后从事软件职业的打算。

参 考 文 献

1. [美]Ron Patton 著. 张小松，王钰，曹跃等译. 软件测试. 第 2 版. 北京：机械工业出版社，2006
2. [美]Glenford J. Myers，Tom Badgett，Todd M. Thomas，Corey Sandler 等著. 王峰，陈杰译. 软件测试的艺术. 北京：机械工业出版社，2006
3. [美]Daniel J. Mosley，Bruce A. Posey 著. 邓波，黄丽娟，曹青春等译. 软件测试自动化. 北京：机械工业出版社，2003
4. 朱少民. 软件测试方法和技术. 北京：清华大学出版社，2005
5. 贺平. 软件测试教程. 北京：电子工业出版社，2005
6. 古乐，史九林. 软件测试技术概论. 北京：清华大学出版社，2004
7. 陆璐，王柏勇. 软件自动化测试技术. 北京：清华大学出版社；北京交通大学出版社，2006
8. 曲朝阳，刘志颖. 软件测试技术. 北京：中国水利水电出版社，2006
9. 李幸超. 实用软件测试：来自硅谷的技术、经验、心得和实例. 北京：电子工业出版社，2006
10. 飞思科技产品研发中心. 实用软件测试方法与应用. 北京：电子工业出版社，2003
11. 罗运模等. 软件能力成熟度模型集成(CMMI). 北京：清华大学出版社，2003
12. 赵瑞莲. 软件测试. 北京：高等教育出版社，2004
13. 张海藩. 软件工程导论. 第 3 版. 北京：清华大学出版社，1998
14. http://www.17testing.com/
15. http://www.cnsoft.cn/
16. http://www.51testing.com/
17. http://www.uml.org.cn/
18. http://www.softtest.cn/
19. http://www.opentest.net/
20. http://www.testage.net/